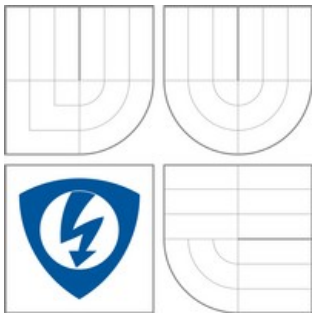
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

BEZDRÁTOVÝ SYSTÉM ŘÍZENÍ OSVĚTLENÍ

LIGHTENING CONTROL SYSTEM WITH WIRELESS INTERFACE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

Jan Hrbáček

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Zdeněk Bradáč, Ph.D.

BRNO 2008

Zadání

Navrhněte koncepci řízení (zapínání, vypínání, stmívání) světel v budovách bez nutnosti instalace trvalých rozvodů. Navrhněte elektroniku řízení světla a ovládací elektroniku tak, aby tyto dvě komponenty mezi sebou komunikovaly bezdrátově (ZigBee/IEEE 802.15.4). Elektroniku vybavte mikrokontroléry. HW realizujte a oživte. Vytvořte nezbytné programové vybavení. Proveďte literární rešerši tématu a obdobných zařízení.

Abstrakt

Bakalářská práce se zabývá návrhem systému bezdrátového řízení osvětlení určeného pro použití v podmínkách nedovolujících instalaci trvalých řídicích rozvodů. Systém sestává z jednoho koordinátorského a více koncových zařízení komunikujících v síti dle standardu IEEE 802.15.4. Součástí práce je rešerše tématu a možných řešení, popis návrhu hardwarového zapojení a mechanické konstrukce a rozbor aplikačního firmware všech navržených druhů zařízení.

Výsledkem práce jsou zařízení realizovaná dle návrhu popsaného v práci. Na základě testování jejich funkčnosti byly učiněny drobné korekce návrhu, finální zařízení jsou plně funkční a spolehlivá.

Klíčová slova: ZigBee, IEEE 802.15.4, řízení osvětlení, bezdrátová komunikace

Abstract

The goal of the bachelor's thesis is to design a lightening control system with wireless interface intended to be used under conditions not allowing permanent control wiring. The system consists of one coordinator and several end devices communicating in accordance with the IEEE 802.15.4 Standard. The thesis includes background research on the theme and possible solutions, description of the circuit and the mechanical construction and analysis of the devices application firmware.

The resulting devices built according to the thesis were continuously improved, so that the design is now fully functional and reliable.

Keywords: ZigBee, IEEE 802.15.4, lightening control, wireless communication

Bibliografická citace

HRBÁČEK, J. *Bezdrátový systém řízení osvětlení.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. 57 s. Vedoucí bakalářské práce Ing. Zdeněk Bradáč, Ph.D.

Čestné prohlášení

„Prohlašuji, že svou bakalářskou práci na téma *„Bezdrátový systém řízení osvětlení“* jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení §11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení §152 trestního zákona č. 140/1961 Sb.“

V Brně dne 28.5.2008

….....

Jan Hrbáček

Poděkování

Na tomto místě bych chtěl poděkovat Ing. Zdeňku Bradáčovi, Ph.D. a Ing. Ondřeji Hynčicovi, Ph.D. za vedení a cenné rady během zpracování této bakalářské práce.

Stejně tak bych chtěl poděkovat své rodině a přátelům za podporu v průběhu celého studia.

Obsah

1. Úvod

ZigBee je relativně nový bezdrátový komunikační standard spadající do kategorie PAN (*Personal Area Network*). Jeho vlastnosti jej předurčují pro nasazení v aplikacích, kde nějakým způsobem nevyhovují ostatní bezdrátové standardy (Bluetooth, WiFi atd.). Vyznačuje se dosahem do 75 metrů a díky jednoduchosti implementace a nízké spotřebě je vhodný zejména pro použití v bateriově napájených zařízeních řízených méně výkonným mikroprocesorem.

Tato práce se zabývá konstrukcí zařízení, využívajícího standard IEEE 802.15.4 pro realizaci komunikace mezi jednou řídicí jednotkou a několika jednotkami podřízenými. Jako modelový případ je vzat systém ovládání osvětlení, neboť právě automatizace budov je jednou z oblastí zamýšleného a výhodného použití této technologie.

Součástí práce je návrh zapojení, řešení mechanické konstrukce jednotek a programové vybavení zajišťující komunikaci odpovídající standardu IEEE 802.15.4 (vzhledem k převládajícím směrům datové komunikace budu dále mluvit o *vysílači* a *přijímači*).

Od *vysílače* je zadáním projektu očekávána možnost výběru ovládaného kanálu (skupiny svítidel), jeho skokové zapnutí/vypnutí a plynulá změna výkonu dodávaného do svítidla. Předpokladem je také bateriový provoz nutný pro přiměřenou mobilitu vysílače.

Přijímač má být navržen a umístěn do elektroinstalační krabice v blízkosti ovládaného svítidla, z čehož vyplývá možnost síťového napájení.

Z důvodu uvedení čtenáře do problematiky standardů ZigBee/IEEE 802.15.4 a volby vhodných hardwarových prostředků obsahuje práce také stručnou rešerši. Zabývám se v ní konstrukčními prvky relevantními zpracovávanému tématu.

2. Rešerše tématu, možná řešení

2.1. Standard ZigBee a IEEE 802.15.4

Nízkorychlostní bezdrátový komunikační standard ZigBee je nová[1] technologie doplňující nabídku na „trhu“ rádiových spojovacích technologií. Oproti ostatním standardům nabízí především jednoduchou implementaci v řídicím mikrokontroléru (např. několikanásobně menší velikost nutného kódu vůči systému Bluetooth), velmi nízkou spotřebu (důraz na možnost bateriového napájení při životnosti běžné baterie až půl roku), relativní obvodovou jednoduchost a z ní vyplývající příznivou cenu.

Samotný standard ZigBee je nadstavba nad specifikací IEEE 802.15.4. Z toho také vyplývá zařazení – pracovní skupina 802.15 se zabývá standardizací prostředků WPAN – *Wireless Personal Area Network*.

2.1.1. Standard IEEE 802.15.4

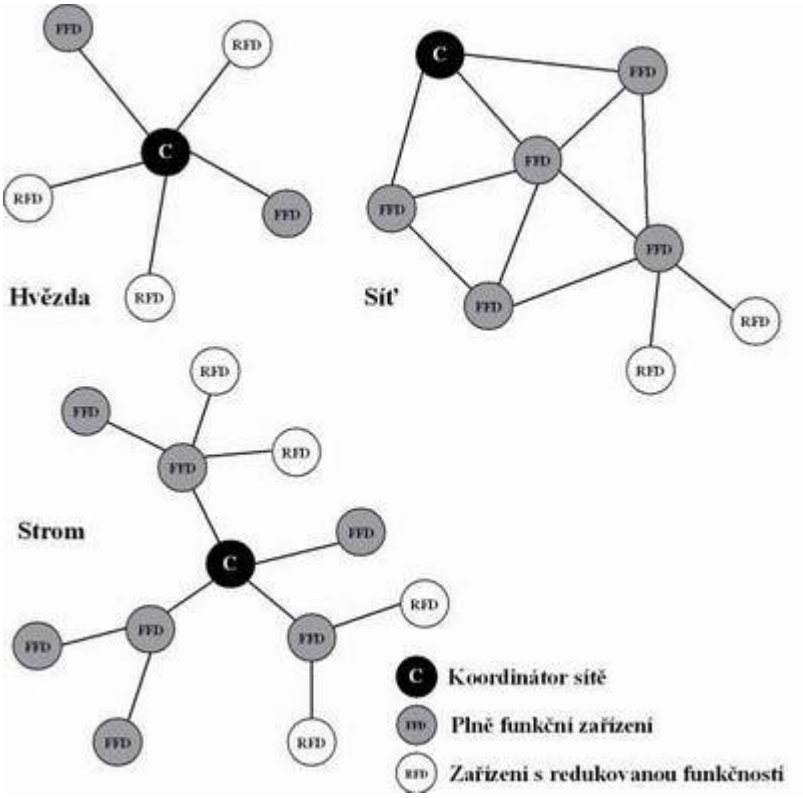
specifikuje fyzickou vrstvu (PHY) a část linkové vrstvy – *Medium Access Control* (MAC).

PHY vrstva se stará o vlastní transceiver, který může dle specifikace fungovat ve třech různých kmitočtových pásmech:

- 868-868.8 MHz: Evropa, původně jeden kanál, rozšířeno na tři (viz [1])
- 902-928 MHz: Severní Amerika, původně deset kanálů, rozšířeno na třicet
- 2400-2483.5 MHz: celosvětově, šestnáct kanálů

Datový signál je na nosnou vlnu modulován metodou OQPSK[2] a posléze rozprostřen pomocí DSSS[3]. Pro přístup k fyzickému médiu se používá metoda CSMA/CA[4], která zajišťuje předcházení kolizím při komunikaci více zařízení současně (viz např. [2]).

1 ZigBee je platným standardem od roku 2004
2 Offset Quadrature Phase-Shift Keying, varianta čtyřstavového fázového klíčování, která má oproti „čisté“ QPSK modulaci do kanálu Q zařazen člen zpožďující o jednu periodu
3 Direct Sequence Spread Signal, rozprostírání spektra pseudonáhodnou posloupností (PNP)
4 Carrier Sense Multiple Access with Collision Avoidance, metoda s vícenásobným přístupem s naslouchánim nosné a předcházením kolizím



Ilustrace 2.1: Různé topologie ZigBee sítí ([3])

Podvrstva MAC specifikuje komunikační protokol založený na datových rámcích. Jsou definovány čtyři typy datových rámců ([4], [5], [6]):

- *Data Frame* – datový rámec, velikost užitečných dat 104 B
- *Acknowledgement Frame* – potvrzovací rámec kontrolovaný na úrovni MAC
- *Beacon Frame* – synchronizační rámec používaný zejména v módu beacon enable, kdy slouží k uvádění zařízení do režimu spánku
- *MAC Command Frame* – typ rámce sloužící pro nastavování a řízení podřízených zařízení v síti ZigBee

Ostatní funkčnost (především síťová) je implementována specifikací ZigBee.

2.1.2. ZigBee

Standard ZigBee zavádí dva druhy zařízení – FFD (*Fully Functional Device*) a RFD (*Reduced Functionality Device*), přičemž RFD mohou pracovat pouze jako koncová zařízení. Koordinátor sítě a směrovače musí být kategorie FFD.

Každé zařízení má podle IEEE 802.15.4 přiřazen binární adresovací kód dlouhý 64 bitů nebo 16 bitů (zkrácený). Sestavená síť je dále identifikována 16ti bitovým PAN ID, které přiděluje koordinátor a slouží k rozlišení překrývajících se sítí.

Topologii sítě lze u ZigBee realizovat třemi způsoby (viz Ilustrace 2.1):

- *hvězda (Star)* – jeden koordinátor, ostatní zařízení pracují jako koncová
- *strom (Tree)* – podobně jako hvězda, avšak koncová zařízení mohou být připojena přes směrovače a lze tak zvětšit vzdálenosti mezi koordinátorem a koncovým zařízením
- *síť (Mesh)* – kombinace předchozích dvou typů topologie, umožňuje realizovat redundanci spojení a zvyšuje tak flexibilitu návrhu sítě a spolehlivost při výpadku funkce některého ze zařízení

2.2. Přehled dostupných platforem

V této části rešerše zmíním některé z vyráběných konstrukčních platforem. Výčet omezím na prvky, které jsou na fakultě používány a jsou tím pro mou práci relevantní. U každého také uvedu vlastnosti (kladné, neutrální a záporné) vyplývající z porovnání platforem mezi sebou.

2.2.1. XBee (Digi International, dříve MaxStream)

Modul osazený mikrokontrolérem Freescale (Series 1) nebo Ember (Series 2).

- ✔ firmware modulů Series 1 podporuje IEEE 802.15.4, Series 2 kompletní ZigBee
- střední rozměry (2,5 x 2,7 cm)

Ilustrace 2.2: Modul Xbee ([7])

Ilustrace 2.3: Modul ZigBit ([8])

Ilustrace 2.4: Modul osazený PAN4450 ([9])

2.2.2. ZigBit (MeshNetics)

Nejspíš vůbec nejmenší modul na trhu s velmi nízkou spotřebou postavený na mikrokontroléru rodiny ATmega.

- ✔ nejmenší rozměry (1,3 x 2,4 cm)
- ● pouze IEEE 802.4.15 stack
- ✗ absence pinheaderů – bez osazení na adaptér nemožnost použít v nepájivém kontaktním poli nebo osadit odnímatelně

2.2.3. PAN4450 a PAN4551 (Panasonic)

Na platformě těchto hybridních čipů jsou založeny moduly vyvinuté na UAMT FEKT VUT.

- ✔ dobrá dostupnost modulů – výrobky domovského ústavu
- ● pouze IEEE 802.14.5 stack
- ✗ největší rozměry cca (4 x 6 cm)

2.3. Napájení

Stabilní a vyhlazené napájení je základem spolehlivé funkce každého elektronického zařízení, v případě rádiových zařízení to platí dvojnásob. Různých možností napájení je nepřeberné množství, já zde rozeberu pouze varianty, které jsou použitelné pro můj projekt.

2.3.1. Baterie a akumulátory

jsou prakticky jedinou volbou pro napájení přenosných přístrojů. Podle vratnosti elektrochemických reakcí se dělí na primární (např. suché články) a sekundární (akumulátory).

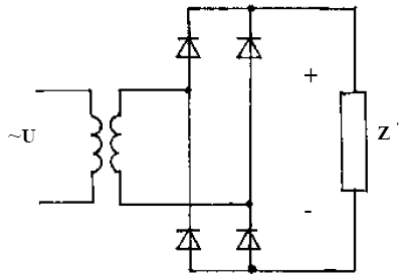
Akumulátory jsou vhodnější pro zařízení, jejichž spotřeba by při použití primárních článků znamenala nutnost jejich časté výměny. Zařízení využívající standard ZigBee však do této kategorie obvykle nespadají, neboť jedním z hlavních přínosů této normy je cílení na nízkou spotřebu energie.

Kvalitní shrnutí včetně výčtu jednotlivých typů primárních i sekundárních článků uvádí pramen [10], uvedení těchto informací zde by bylo poněkud nad rámec zpracovávaného tématu.

2.3.2. Galvanicky oddělený napájecí zdroj

Tímto zdrojem je myšleno zapojení zdroje stejnosměrného zdroje napětí složené z těchto základních prvků:

- transformátor – počet vinutí je určen typem použitého usměrňovače (dvoupulsní uzlový usměrňovač vyžaduje vyvedený střed vinutí)



Ilustrace 2.5: Typické zapojení galvanicky izolovaného zdroje ([11])

- usměrňovač – nejčastěji používaný je dvoupulsní můstkový
- filtrace, vyhlazování – u jednoduchých konstrukcí je obvykle dostačující paralelně připojený kondenzátor
- stabilizátor – nejčastěji lineární, na energetické ztrátě způsobené úbytkem napětí na stabilizátoru při napájení z rozvodné sítě většinou nesejde

Pokud zadání neklade požadavky na velikost výsledného zařízení, bývá galvanicky izolovaný napájecí zdroj dobrou volbou, zejména pro svou jednoduchou konstrukci a oddělenost od rozvodné sítě (což zjednodušuje měření prováděná na obvodu).

2.3.3. Neoddělený napájecí zdroj

Oproti předchozímu zapojení postrádá tato konstrukce zdroje transformátor. Síťové napětí je zde přivedeno na stabilizační Zenerovu diodu přes sériově zapojený kondenzátor, který bezztrátově omezuje proud jí tekoucí. Projevuje se totiž jeho kapacitance daná vztahem

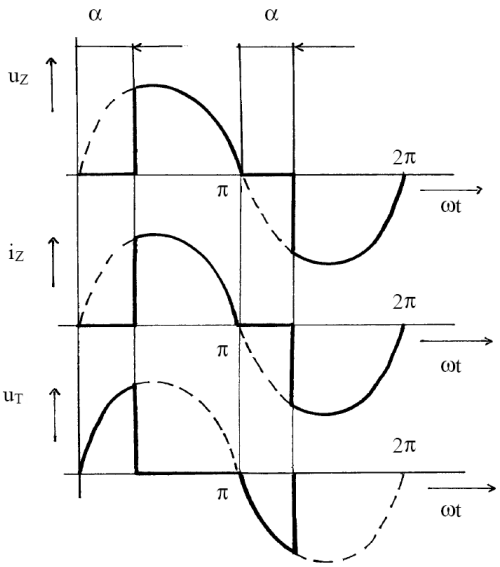
$$X_C = \frac{1}{j\,\omega\,C} = \frac{1}{j\,2\pi\,f\,C}$$

Velikost kapacitance je tedy nepřímo úměrná kapacitě a frekvenci.

Za touto napěťovou stabilizací následuje opět filtrace, nejčastěji pomocí kondenzátorů, případně pomocí tlumivky.

2.4. Fázové řízení spotřebiče

Fázové řízení je metoda řízení výkonu dodávaného spotřebiči ze střídavé napájecí sítě. Jde ve zkratce o to, že po průchodu síťového napětí nulou se o určitou chvíli pozdrží zapnutí výkonového prvku (obvykle triaku – musí to být prvek, který se sám rozepne při opětovném průchodu napětí nulou), čímž se zkrátí perioda, během které je zátěž připojena k síti. Tato prodleva je dána *řídicím úhlem* α, což je úhel v rozsahu od 0 do π. Platí, že čím větší je řídicí úhel, tím později je zátěž připojena a tím menší je dodávaný výkon.



Ilustrace 2.6: Průběhy napětí a proudu na R zátěži a napětí na triaku ([11])

2.5. Vstupní zařízení

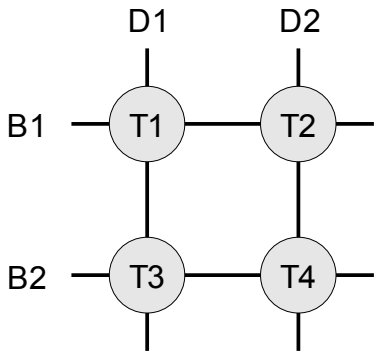
Je-li součástí zadání konstrukce libovolného přístroje interakce s uživatelem, zpravidla se neobejdeme bez některého ze vstupních zařízení. Nejjednodušším případem takové periferie je spínač nebo vypínač přímo připojený ke vstupnímu pinu mikrokontroléru. Vzhledem k tomu, že počet vstupních pinů není nekonečný (dvojnásob to platí u pinů schopných vyvolat přerušení) a je třeba jimi šetřit, jsou často používána maticová uspořádání klávesnic.

Použití tlačítek je však v některých aplikacích vzhledem k uživatelskému komfortu minimálně nevhodné. Přijatelnou alternativu pak tvoří různé otočné či posuvné voliče absolutního i inkrementálního[5] charakteru.

5 Příkladem absolutního snímače může být potenciometr či snímač s Grayovým kódem, inkrementálním charakterem se vyznačuje např. scrollovací prvek počítačové myši.

2.5.1. Maticová klávesnice

Jak již název napovídá, jsou tlačítka uspořádána do formy matice s řádky označenými B (buzení) a sloupci D (detekce). Stisknutí tlačítka T způsobí propojení budicí a detekční linky.



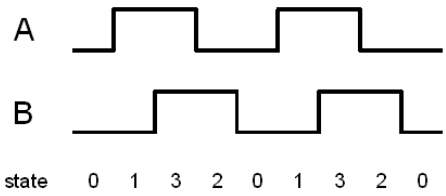
Ilustrace 2.7: Princip maticové klávesnice

Za klidového stavu jsou zpravidla buzeny všechny řádky. Stisk libovolného tlačítka tak může např. vyvolat přerušení a probudit mikrokontrolér ze stavu snížené spotřeby.

Rutina provádějící obsluhu klávesnice pak (obvykle v cyklu) postupuje po jednotlivých řádcích, které samostatně aktivuje, a čte signál přicházející ze sloupců. Je tak schopná zjistit souřadnice „zkratu“ v matici.

2.5.2. Scroller s kvadraturním výstupem

Úroveň jasu svítidla je vhodné ovládat inkrementálním způsobem – tedy posílat příkazy na změnu jasu o určitý krok, nikoliv na určitou hodnotu. Jako ideální a intuitivní se v tomto kontextu jeví použití ovládacího prvku známého z počítačové myši – tedy scrollovacího kolečka. Jde zpravidla o snímač pracující na optickém nebo mechanickém principu a na výstupu poskytující signál kvadraturního charakteru (dvě složky vzájemně fázově posunuté).



Ilustrace 2.8: Složky signálu IRC

Ilustrace 2.8 uvádí příklad takového signálu. V rámci jedné periody jej můžeme rozdělit na čtyři úseky, kdy každému z těchto úseků je přiřazeno číselné označení. Zdánlivě nelogická posloupnost čísel se ozřejmí, pokud na číslo nahlížíme jako na dvoubitové slovo BA, kde písmena A a B zastupují logickou hodnotu jednotlivých složek kvadraturního signálu.

Z posloupnosti takto označených stavů již není problém určit směr otáčení.

3. Volba koncepce zařízení

Otázka vhodné volby struktury sítě a typů v ní komunikujících zařízení se na první pohled zdá poměrně zřejmá: v rámci jedné sítě se předpokládá použití jednoho vysílače a několika přijímačů, což svádí k ustanovení vysílače koordinátorem a přijímače koncovým zařízením. Avšak uvědomíme-li si, že vysílač je navrhován jako zařízení snadno přenosné a tedy nejspíš bateriově napájené, vyvstane první problém – koordinátor by měl být mimo vlastní komunikaci stále na příjmu (a z tohoto důvodu napájen z rozvodné sítě).

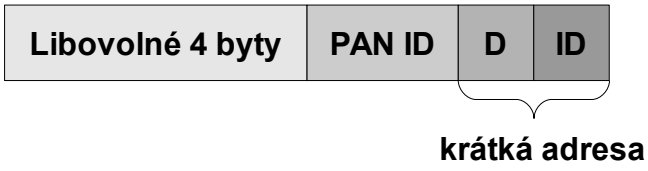
Je proto třeba definovat tři druhy zařízení podílející se na funkci sítě:

- *vysílač* – koncové zařízení, určené pro interakci s uživatelem
- *přijímač-koordinátor* – centrální zařízení sítě; vytváří síť a přijímá asociace ostatních zařízení; také výkonový člen
- *koncový přijímač* – výkonový člen určený pro řízení konkrétního svítidla

Po stránce hardwaru jsou oba druhy přijímače totožné, jejich rozdílná funkčnost tkví pouze v nahraném firmware.

Druh zařízení jsem se rozhodl reflektovat v jeho MAC adrese (potažmo i v krátké adrese přidělované koordinátorem). Mimo této informace obsahuje rozšířená adresa zařízení také identifikátor sítě PAN (viz Ilustrace 3.1).

Význam identifikátoru zařízení o délce jednoho bytu je také závislý na druhu zařízení – zatímco v případě vysílače má celý tento byte význam jednoho identifikátoru, přijímače jím rozlišují také číslo kanálu (horní 4 bity). Je tak možné v rámci jedné sítě PAN adresovat až 256 vysílačů a 256 koncových přijímačů.



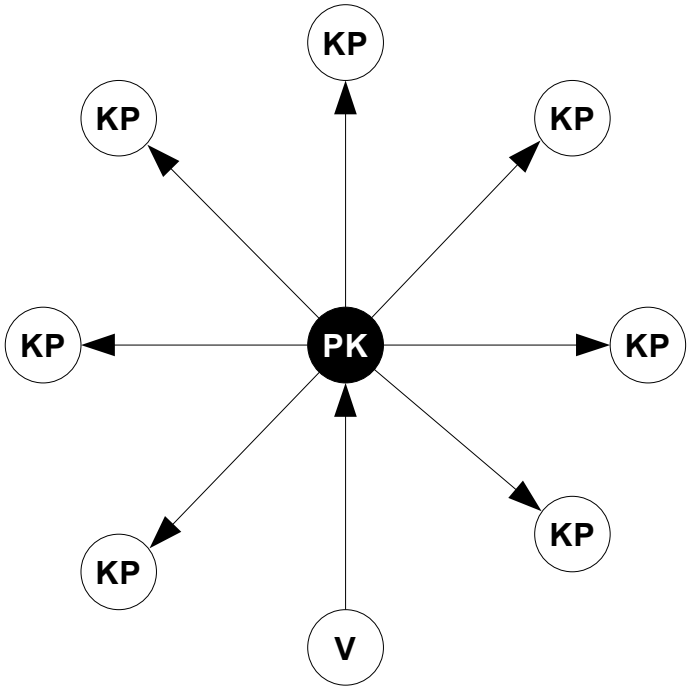
Ilustrace 3.1: Struktura MAC adresy (D - druh zařízení, ID - identifikátor zařízení)

V rámci jednoho kanálu (kterých je 16) je pak možné současně řídit až 16 koncových přijímačů (v jednom z kanálů může navíc figurovat i přijímač-koordinátor).

Topologie sítě je typu hvězda (viz Ilustrace 3.2). Všechny přenosy dat jsou přímé (*Direct Data Transfer*).

Komunikace pak probíhá podle následujícího schématu:

1. *Vysílač* na požadavek uživatele vytvoří příslušný aplikační příkaz a odešle jej *koordinátoru*.
2. *Koordinátor* z příkazu přečte identifikátor kanálu, kterému je příkaz adresován. Je-li to kanál, na nějž je sám „připojen“, předá zprávu vlastní vyhodnocovací logice.
3. Příkaz předá *koordinátor* také odesílací rutině, která projde seznam asociovaných *koncových přijímačů* pro příslušný kanál a všem nalezeným zařízením příkaz odešle.



Ilustrace 3.2: Typická topologie sítě (PK - přijímač-koordinátor, KP - koncový přijímač, V - vysílač)

3.1. Komunikační protokol

Pro jednoznačnou komunikaci mezi jednotlivými zařízeními je nutné ustanovit interní aplikační protokol.

Požadavky vyplývající z charakteru úlohy jsou poměrně jasné – příkaz musí nést informaci o ovládaném kanálu, žádané operaci a případně i parametr operace. Pro všechny tyto tři informace dostačuje prostor jednoho bytu, celý příkaz tedy bude mít délku tří bytů. Pořadí informací je pak následující:

```
<kanál> <identifikátor příkazu> <parametr příkazu>
```

Z důvodu jednoduché a přehledné práce s příkazem v rámci aplikačního kódu byl definován typ `appMessage_t`, který označuje strukturu obsahující samostatný prvek pro každou z uvedených informací (viz kapitolu 5.1. Společné části kódu).

V současné době aplikace podporuje tyto čtyři příkazy:

- **`0x01`** – zvýšit jas svítidla. Parametr v procentech udává, o jak velký skok má být jas navýšen.
- **`0x02`** – snížit jas svítidla. Parametr má stejný význam jako v případě předchozího příkazu.
- **`0x03`** – zapnout svítidlo na plný výkon. Parametr může být libovolný, není při interpretaci příkazu brán v potaz.
- **`0x04`** – vypnout svítidlo. Parametr může opět nabývat libovolné hodnoty.

4. Návrh hardware

Tato kapitola popisuje hardware obou částí stmívače. Jeho větší část byla navržena v rámci semestrálního projektu, od té doby však byly provedeny některé dílčí změny. Celková schémata a navržené desky plošných spojů jsou k dispozici v příloze a na CD, zde jsou podrobně rozebrány jednotlivé části obou zapojení.

Návrh schémat i desek plošných spojů byl realizován za pomoci programového balíku *Eagle 5.0 Light* a je k dispozici v kompletní formě v obrazové příloze práce.

4.1. Vysílač

V souladu s požadavky je vysílač konstruován jako ovladač většího počtu samostatných přijímačů-stmívačů pracujících na různých nebo i stejných kanálech (kanálem je v tomto kontextu myšlena adresa přijímače – vzhledem k tomu, že může pracovat více zařízení se stejnou adresou, přijde mi vhodnější použití termínu kanál, neboť pojem adresa bývá obvykle chápán jako unikátní identifikace daného zařízení).

4.1.1. Napájení

Vysílač je koncipován jako samostatné přenosné zařízení, je tedy napájeno bateriemi. Při volbě vhodného zdroje jsem bral v potaz především minimalizaci ztrát, které by zbytečně zkracovaly dobu životnosti baterií. Zavrhl jsem proto již na počátku použití lineárního stabilizátoru. Energeticky výhodnější by byl spínaný měnič, ať už typu step-up nebo step-down (realizovaný např. známým obvodem MC34063). Nevýhodou je, že obvykle potřebují určitý minimální zatěžovací proud k tomu, aby poskytovaly stabilní nezvlněné výstupní napětí, což by v našem případě, kdy je třeba se snažit o minimální odběr, nebylo vhodné. Nakonec jsem se tedy rozhodl napájet vysílač nestabilizovaným napětím 3 V přímo z baterie dvou článků velikosti AA nebo AAA – napájecí napětí hybridního čipu má být v rozmezí 2,1 až 3,4 V, což použití takovéhoto napájení dobře umožňuje.

Výhodou zvoleného řešení je fakt, že při uvedení řídicího procesoru do spánku

klesne klidový proudový odběr na úroveň desetin až jednotek mikroampér, což by při použití jakéhokoliv stabilizátoru nebylo možné.

4.1.2. Řídicí a komunikační část

„Mozkem“ celého zařízení je již zmíněný modul osazený hybridním čipem Panasonic PAN4551. Součástí hybridního čipu je mikrokontrolér rodiny Freescale HCS08, který plní úlohu řídicího prvku celé konstrukce.

Jedinou periferií připojenou k modulu je několik skupin ovládacích tlačítek.

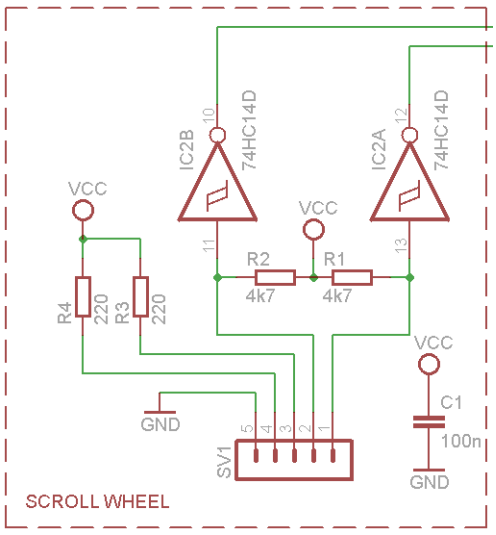
Veškeré obvody potřebné pro správnou funkci čipu jsou již realizovány na plošném spoji modulu, stejně tak i programovací konektor. Nebudu se jimi zde proto zabývat.

4.1.2.1. „Uživatelské rozhraní“

Ovládání vysílače je tvořeno dvěma částmi – tlačítky pro výběr řízeného kanálu stmívače a prvky nastavení úrovně osvětlení.

K výběru kanálu slouží osm tlačítek, kterým mohou být programově přiřazeny jak přímo konkrétní kanály, tak i jiné funkce pro práci s výběrem kanálů (např. volba všech kanálů atd.).

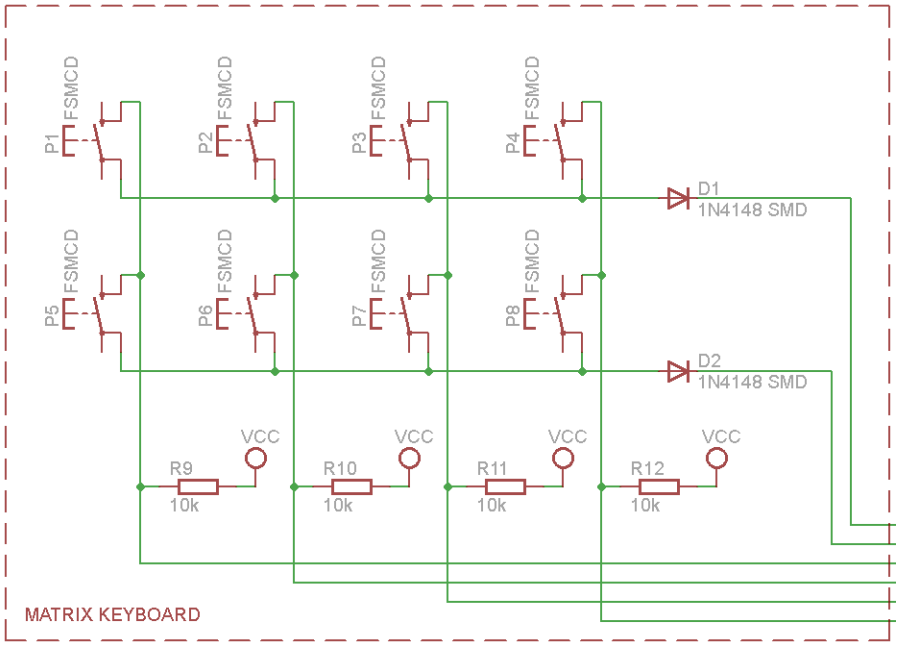
Samotné řízení zvoleného kanálu obstarává dvojice tlačítek pro skokové zapnutí a vypnutí ovládaného spotřebiče doplněná o scrollovací kolečko z myši určené pro plynulou změnu výkonu. Toto pracuje na principu optického kvadraturního snímače. Pro spolehlivé rozpoznávání hran a úrovní IRC signálu je mezi mikrokontrolér a kolečko zapojen tvarovač tvořený Schmittovými KO (Ilustrace 4.1).



Ilustrace 4.1: Tvarovač IRC impulsů scrolleru

Vzhledem k tomu, že kvůli úspoře napájecích zdrojů byl měl být procesor většinu času v režimu spánku, je třeba, aby

všechna tlačítka byla schopna vyvolat při stisku přerušení procesoru a probudit jej z tohoto režimu. Zabudovaný mikrokontrolér rodiny HCS08 disponuje modulem KBI (*Keyboard Interrupt*), který je uvedeného chování přesně schopen. Bohužel je však omezen na piny portu A, tedy osm pinů, zatímco je potřeba obsluhovat vstupy z deseti tlačítek a jednoho IRC.



Ilustrace 4.2: Schéma maticové klávesnice volby kanálu

Tuto situaci jsem vyřešil použitím maticové klávesnice (viz Ilustrace 4.2) pro realizaci voliče kanálů a upotřebením zbylých čtyř pinů pro obsluhu IRC a dvou tlačítek zapnout/vypnout. Pro správnou funkci bude na společných vodičích sloupců maticové klávesnice v režimu spánku ponechán potenciál záporného pólu napájení, čímž dojde při stisku kteréhokoliv tlačítka k vyvolání přerušení a probuzení mikrokontroléru z režimu spánku.

4.1.3. Mechanické řešení, návrh DPS

Prototyp vysílače je určen pro zástavbu do plastové krabičky KPDO03 o rozměrech 27 x 59 x 188 mm a podlouhlém tvaru dálkového ovladače. Krabička má oddělený prostor vyhrazený pro 9V baterii – tento jsem využil pro umístění dvou článků velikosti AAA, které jsou 9V baterii rozměrově nejblíže.

Je počítáno s vyvrtáním otvorů pro tlačítka a scroller v horní části krabičky, které pak budou zakryty fólií s potiskem.

Ilustrace 4.3: KPDO03 ([12])

Jednou ze složitějších částí mechanické stavby vysílače je zabudování scrollovacího kolečka. Vzhledem k tomu, že jde o prvek získaný z běžné počítačové myši, není jeho konstrukce uzpůsobena snadné demontáži a montáži v jiném zařízení. Upevnění kolečka je tedy otázkou improvizace pro každý konkrétní typ myši.

4.2. Přijímač

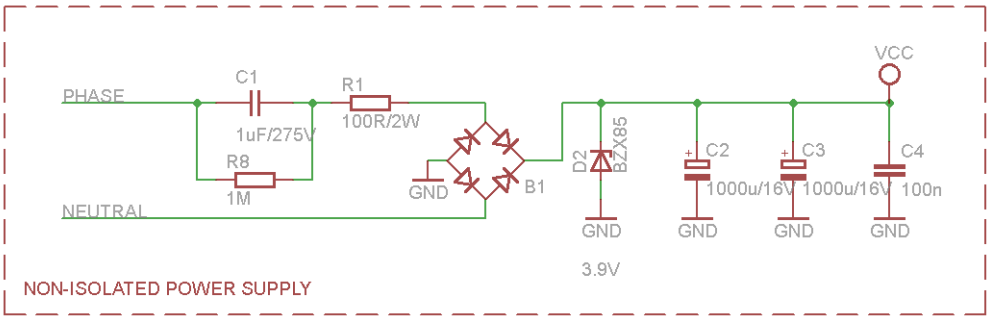
V rámci této práce je navržen jednovýstupový stmívací modul schopný fázově řídit výkon běžné žárovky. Stmívání zářivek je velice komplexní problém a univerzální řešení prakticky neexistuje. Proto jsem se rozhodl vůbec jej neřešit; případná potřeba dálkového stmívání zářivky by pak byla nejspíš řešena použitím některého z průmyslově vyráběných řídicích obvodů určeného pro konkrétní typ zářivky a na něj navázané bezdrátové jednotky.

Konstrukce přijímače je v podstatě jen demonstrací; v praxi by byl nejspíš výhodnější návrh vícečetné stmívací jednotky, která by lépe rozložila cenu komunikačního ZigBee modulu mezi více ovládaných světel. Takovéto rozšíření by nepřineslo potřebu prakticky žádných nových konstrukčních prvků – nejjednodušším řešením by bylo prostě „naklonovat“ výkonovou sekci přijímače.

4.2.1. Napájení

Plánované použití přijímače spočívá v zabudování do instalační krabice poblíž řízeného svítidla. K dispozici tedy bude po celou dobu elektrická napájecí síť; výpadky sítě řešit nemusíme, protože v době výpadku stejně „není co řídit“.

Zdroj je řešen zejména z prostorových důvodů jako galvanicky neizolovaný od sítě, nepotřebuje tedy prostorově náročný transformátor. Zapojení je standardní, je použit již zmíněný dvoupulsní můstkový usměrňovač, na jehož výstupu je zapojena stabilizační Zenerova dioda a filtrační a vyhlazovací kondenzátory.



Ilustrace 4.4: Schéma napájecího zdroje přijímače

Sériový kondenzátor C1 zajišťuje omezení proudu protékajícího Zenerovou diodou, potažmo napájeným obvodem. Musí být dimenzován na dostatečně vysoké napětí – zvolil jsem fóliový typ X2 pro nominální střídavé napětí 275 V. Jeho kapacita je zvolena tak, aby kapacitní reaktancí omezoval proud na cca 73 mA:

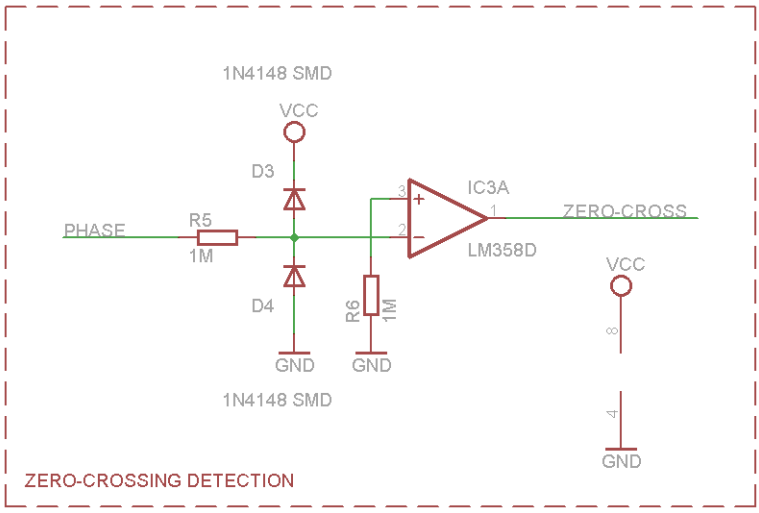
$$I = \frac{U}{|X_C|} = U \cdot \omega \cdot C = 230 \cdot 100\,\pi \cdot 1000 \cdot 10^{-9} = 73\,\text{mA}$$

Ke zdroji jsou dále připojeny vyhlazovací elektrolytické kondenzátory, které pomohou pokrýt případné proudové špičky. Návrh zdroje počítá s osazením lineárního low-drop stabilizátoru na desce modulu s čipem PAN4551, který napětí stabilizované Zenerovou diodou upraví na výsledné napětí použité pro napájení hybridního čipu.

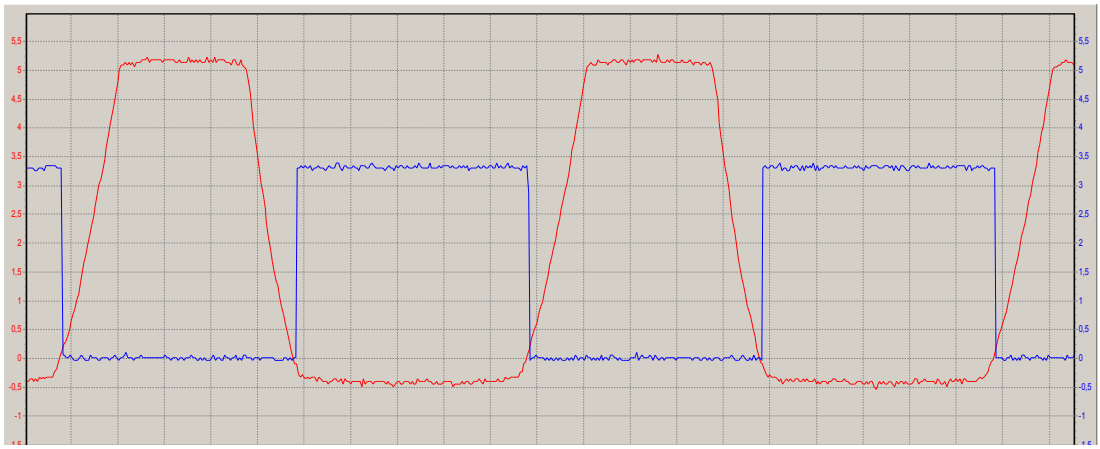
4.2.2. Detektor průchodu nulou

Již bylo nastíněno, že výkon svítidla je řízen fázově. Princip je popsán v kapitole věnující se rešerši tématu.

Z něj je mimo jiné patrné, že je třeba detekovat průchod síťového napětí nulou. O to se stará detektor, jehož schéma naleznete na obrázku 4.5. Jeho základ tvoří proudový omezovací rezistor R5 následovaný omezovačem napětí tvořeným diodami D3 a D4. Ty omezí amplitudu napětí na interval přibližně (-0,5 V; Vcc + 0,5 V). Signál je následně tvarován operačním zesilovačem IC2, který pracuje v režimu komparátoru. Jako referenční potenciál je rezistorem R6 přiveden potenciál nulového vodiče. Na výstupu komparátoru je pak obdélníkový signál o úrovni logické nuly při kladné velikosti síťového napětí a logické jedničky při síťovém napětí záporném.



Ilustrace 4.5: Obvod detekce průchodu síťového napětí nulou



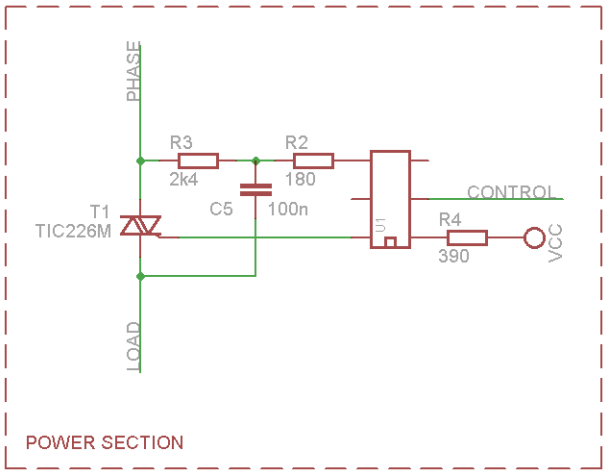
Ilustrace 4.6: Průběh napětí na invertujícím vstupu (červená křivka) a výstupu OZ

Ilustrace 4.6 zachycuje popsanou situaci. Tento průběh byl naměřen na testovacím zapojení detektoru realizovaném na nepájivém kontaktním poli, na jehož vstup bylo z důvodu snadného měření přivedeno transformované napětí o velikosti cca 20 V. V praxi budou oproti tomuto modelu výsledky ještě poněkud lepší (tzn. menší zpoždění mezi průchodem síťového napětí nulou a překlopením komparátoru), protože díky asi desetinásobné velikosti síťového napětí vzroste strmost hran „ořezaného“ průběhu.

4.2.3. Výkonová část

Zapojení výkonové sekce přijímače je na obrázku 4.7. Tvoří ji výkonový triak T1, k jehož ovládání slouží optotriak U1 v katalogovém zapojení ([13]). Rezistor R3

a kondenzátor C5 jsou příprava pro řízení zátěže induktivního charakteru, při použití s odporovou zátěží se C5 neosadí a R3 propájí.



Ilustrace 4.7: Výkonová část přijímače

Triak TIC226M je dimenzován na napětí 600 V a trvalý proud 8 A ([14]). Pro ovládání běžných svítidel tedy bude s rezervou dostačovat.

4.2.4. Řídicí a komunikační část

Přijímač je stejně jako vysílač založen na modulu osazeném hybridním čipem Panasonic PAN4551. Tento zabezpečuje jak komunikaci s vysílačem, tak i řízení triaku a tím i samotného svítidla.

Pro přiměřeně přesné fázové řízení je velmi důležité načítání dat z detektoru průchodu síťového napětí nulou. Z osciloskopických průběhů napětí na detektoru (Ilustrace 4.7) je patrné, že událost průchodu nulou je signalizována překlopením výstupu komparátoru do opačné logické/napěťové úrovně. Pro řídicí mikrokontrolér není problém signál tohoto typu dekódovat vyvoláním přerušení při každé změně úrovně na vstupním pinu. Pro tuto potřebu byl vybrán pin PTA7, který je součástí portu A implementujícího modul KBI mikrokontroléru. Horní čtyři piny (tedy i PTA7) tohoto portu jsou schopny detekovat jak sestupnou, tak i náběžnou hranu signálu, plně tedy zamýšlenému účelu vyhovují.

Výkonová část stmívače je řízena impulsy z pinu PTD2, jinak značeného TPM1CH2. Jde o I/O pin, který je nakonfigurovatelný jako výstup druhého kanálu

časovače TPM1. Předpokládám totiž použití *Output Compare* módu tohoto časovače pro realizaci fázové prodlevy po průchodu síťového napětí nulou.

Posledním „zařízením“, které je mikrokontrolérem řízeno, je LED dioda použitelná pro libovolnou signalizaci provozního stavu přijímače.

4.2.5. Mechanické řešení, návrh DPS

Jak již bylo zmíněno, přijímač má být umístěn do instalační krabice v blízkosti ovládaného světla. Je tomu samozřejmě přizpůsoben návrh plošného spoje. Pro konstrukci prototypu jsem vybral standardní čtvercovou plastovou krabici LK 80x28, do které se ZigBee modul prakticky přesně vejde.

Ilustrace 4.8: Krabička LK 80x28 ([15])

Plošný spoje je navržen jako jednostranný, aby jej bylo možné jednoduše vyrobit i svépomocí[6].

Stejně jako na plošném spoji vysílače, i zde jsem jako rezervu pro případné provizorní rozšíření funkčnosti navrhl pole pájecích plošek s vyvedeným napájením a zemí.

Podle konkrétních potřeb při případném nasazení systému není problém návrh upravit např. pro zástavbu do jiné krabičky.

6 Například metodou nažehlení toneru laserové tiskárny, která při správném provedení dává takřka dokonalý výsledek. Speciální fólii pro přenos toneru prodává např. GES Electronics.

5. Programové vybavení

Komunikace podle standardu IEEE 802.15.4 je velmi komplexní záležitost. Zařízení, které se jí má účastnit, musí splňovat jak hardwarové parametry (ošetřené v našem případě hybridním čipem Panasonic PAN4551), tak náležitosti týkající se komunikačního protokolu, řízení provozu v síti atd., které jsou řízeny výhradně softwarově. Je proto evidentní, že pro efektivní vývoj takovýchto zařízení jsou nezbytně nutné kvalitní knihovny implementující nejlépe vrstvy PHY i MAC.

Pro použití s vlastními mikrokontroléry a transceivery poskytuje firma Freescale knihovnu *802.15.4 MAC PHY Software* ([16]), která zmíněnou funkčnost implementuje. Je použita verze 1.0.63.

V dokumentu *802.15.4 MAC PHY Software Reference Manual* ([16]) se uvádí: *„Freescale strongly recommends that users base their application development on the* Freescale 802.15.4 MAC/PHY My_Wireless_App_demo *application example software.“*, volně přeloženo jako *„Freescale zákazníkům důrazně doporučuje založit své aplikace na demo projektu* My_Wireless_App_demo.*“*. Vzhledem k tomu jsem základ firmware odvodil od doporučovaného dema, konkrétně od jeho podverze *MyApp_Ex05* implementující základní rutiny pro komunikaci bez *Beacon Frames*.

Funkčnost zvoleného demoprojektu popisuje dokument *802.15.4 MAC PHY Software User's Guide* ([17]), který také přehledně shrnuje operace nutné např. pro vytvoření nové sítě, asociaci zařízení do existující sítě apod.

Firmware jednotlivých typů zařízení obsahuje části společné všem typům, popíšu je proto samostatně.

5.1. Společné části kódu

Na začátku zdrojových souborů aplikace musí být vloženy hlavičkové soubory knihovny *802.15.4 MAC PHY*:

```
#include "802_15_4.h"
#include "ToolBox.h"
```

Soubor `802_15_4.h` zajišťuje vložení všech potřebných hlavičkových souborů

rozhraní knihovny a definici některých konstant. V souboru `ToolBox.h` jsou pak obsaženy definice některých maker pro práci s pamětí.

Následuje definice maker a konstant použitých v aplikačním kódu:

- `SCAN_CHANNELS` – určuje, které kanály mají být zahrnuty do scanu
- `MAX_PENDING_DATA_PACKETS` – maximální velikost fronty datových paketů
- `DEFAULT_DATA_LENGTH` – délka užitečných dat použitá pro alokaci paketu
- `TX_BUFFER_LEN`, `RX_BUFFER_LEN` – velikost kruhového bufferu vysílaných a přijímaných dat
- `USE_UART` – definicí tohoto makra povolíme ladicí výstup na sériový port
- makra použitá pro přístup k HW – signalizační LED a I/O pinům procesoru

Dále je třeba definovat vlastní typ – strukturu aplikačního příkazu:

```
typedef struct appMessage_tag {
    uint8_t channelId;
    uint8_t command;
    uint8_t data;
} appMessage_t;
```

Je zřejmé, že příkaz obsahuje identifikátor kanálu, jemuž je adresován, vlastní instrukci a parametr instrukce.

Za dopřednou deklarací funkcí následuje definice jednotlivých stavů stavového automatu řídicího funkci zařízení, návratových kódů aplikačních rutin a druhů zařízení účastnících se komunikace v síti (viz kapitolu 3. Volba koncepce zařízení).

Úvodní část kódu končí deklarací globálních proměnných:

- `uint8_t state` – aktuální stav stavového automatu aplikace
- `const uint8_t panId[2]` – identifikátor sítě, nastavený při inicializaci na hodnotu získanou z rozšířené adresy
- `nwkToMcpsMessage_t *pPacket` – ukazatel na strukturu datového paketu
- `uint8_t msduHandle` – unikátní identifikátor paketu
- `uint8_t numPendingPackets` – čítač paketů v odchozí frontě

- `anchor_t mMlmeNwkInputQueue, mMcpsNwkInputQueue` – fronty zpráv rozhraní MAC příkazů a dat
- `appMessage_t *txBuffer[], *rxBuffer[]` – staticky definované kruhové buffery vysílaných a přijímaných dat
- `uint8_t txBufferHead, txBufferTail, rxBufferHead, rxBufferTail` – „ukazatele“ na počátek a konec dat v kruhových bufferech

5.1.1. Řízení běhu programu

Celá logika aplikačního stavového automatu je implementována v rámci „nekonečného“[7] cyklu ve funkci `main`. Před spuštěním tohoto cyklu je třeba definovat lokální proměnné a inicializovat fronty zpráv:

```
void *pMsgIn;
uint8_t rc;
uint8_t macStatus;

/* Initialize variables */
state = stateInit;

/* Prepare input queues.*/
MSG_InitQueue(&mMlmeNwkInputQueue);
MSG_InitQueue(&mMcpsNwkInputQueue);
```

Ukazatel `pMsgIn` je používán pro práci se zprávou přijatou v aktuální iteraci cyklu, do proměnné `rc` je ukládána návratová hodnota některých funkcí a do proměnné `macStatus` stav sítě vracený funkcí `Mlme_Main`.

Stavový automat je inicializován stavem `stateInit` a proměnným `mMlmeNwkInputQueue` a `mMcpsNwkInputQueue` jsou přiřazeny příslušné fronty.

V každé iteraci cyklu je nejprve inicializována proměnná `rc` a učiněn pokus o přečtení potenciálně přijaté zprávy:

```
rc = errorNoError;

/* Try to get a message from MLME */
if (MSG_Pending(&mMlmeNwkInputQueue))
     pMsgIn = MSG_DeQueue(&mMlmeNwkInputQueue);
else
     pMsgIn = NULL;
```

7 Tento cyklus běží, dokud se aplikace nedostane do stavu `stateTerminate` – pak funkci ukončí. Aplikace by se do tohoto stavu neměla dostat.

Posléze je kód rozvětven podle aktuálního stavu. V inicializačním stavu `stateInit` jsou spuštěny rutiny každé z použitých knihoven a rutina nastavující periferie mikrokontroléru (musí být volána až po funkci `Init_802_15_4`):

```
#ifdef USE_UART
     Uart_Init();
#endif
     /* Initialize the 802.15.4 stack */
     Init_802_15_4();
     /* Initialize the rest of the HW */
     HW_init();
     /* Initialize addresses */
     Addr_init();
```

Volání inicializační rutiny knihovny obsluhující UART je, jak již bylo zmíněno, podmíněno definicí makra `USE_UART`.

Ostatní aplikační stavy jsou rozdílné pro koordinátor a koncové zařízení a jsou popsány ve specializovaných kapitolách.

5.1.2. Obsluha rozhraní knihovny 802.15.4 MAC PHY

Následující funkce obsluhy zpráv mezi knihovnou a aplikací jsou převzaty z demoprojektu – starají se o umístění přijatých zpráv do příslušných front.

Rozhraní MLME slouží k předávání systémových zpráv (např. požadavek na provedení aktivního/pasivního scanu, asociaci do sítě atd.):

```
uint8_t MLME_NWK_SapHandler(nwkMessage_t *pMsg)
{
     MSG_Queue(&mMlmeNwkInputQueue, pMsg);
     return gSuccess_c;
}
```

Rozhraní MCPS je rozhraním datovým a má svou vlastní frontu:

```
uint8_t MCPS_NWK_SapHandler(mcpsToNwkMessage_t *pMsg)
{
     MSG_Queue(&mMcpsNwkInputQueue, pMsg);
     return gSuccess_c;
}
```

Zprávy od vrstvy ASP jsou pouze uvolněny z paměti, není třeba se jimi zabývat:

```
uint8_t ASP_APP_SapHandler(aspToAppMsg_t *pMsg)
{
     MSG_Free(pMsg);
     return gSuccess_c;
}
```

5.1.3. Obsluha front aplikačních příkazů

Data určená k odeslání i data přijatá ze sítě jsou skladována v kruhových bufferech `txBuffer` a `rxBuffer`. O jejich obsluhu se starají pomocné funkce `setTransmitData`, `getTransmitData`, `setReceiveData` a `getReceiveData` inspirované knihovními funkcemi obsluhy sériového rozhraní mikrokontroléru. Jejich plné znění naleznete na přiloženém CD.

Funkce **`setTransmitData`** pouze uloží ukazatel na dynamicky alokovanou strukturu příkazu typu `appMessage_t` do kruhového odesílacího bufferu na aktuální pozici čela bufferu. Tuto posléze inkrementuje a inkrementovanou hodnotu logicky vynásobí s maskou bufferu:

```
void setTransmitData(appMessage_t *pMsg)
{
     txBuffer[txBufferHead] = pMsg;
     txBufferHead = (txBufferHead + 1) & (TX_BUFFER_LEN - 1);
}
```

Funkce **`getTransmitData`** nejprve zkontroluje, zda jsou v bufferu k dispozici data. Pokud ano, uloží ukazatel na dynamicky alokovanou strukturu příkazu do pomocné proměnné `pMsg` a posune konec bufferu o jednu (právě uvolněnou) pozici:

```
pMsg = txBuffer[txBufferTail];
txBufferTail = (txBufferTail + 1) & (TX_BUFFER_LEN - 1);
```

Následně jsou data příkazu překopírována do odesílacího bufferu daného ukazatelem `pBuffer` a aplikační příkaz je uvolněn z paměti. Funkce vrací počet bytů určených k odeslání (fixně 3 byty na jeden příkaz):

```
// copy the message to the buffer
*(pBuffer++) = pMsg->channelId;
*(pBuffer++) = pMsg->command;
*(pBuffer++) = pMsg->data;

MSG_Free(pMsg);
pMsg = NULL;

return 3;
```

Funkce **`setReceiveData`** přebírá data z přijímacího bufferu datového rozhraní MCPS. Pokud jsou indikována data o délce menší než 3 byty, je funkce ukončena.

V opačném případě se pokračuje alokováním nového příkazu typu `appMessage_t`, jeho naplněním daty a zařazením do přijímacího kruhového bufferu:

```
pMsg = MSG_AllocType(appMessage_t);

// copy the message to the buffer
pMsg->channelId = *(pData++);
pMsg->command = *(pData++);
pMsg->data = *(pData++);

rxBuffer[rxBufferHead] = pMsg;
rxBufferHead = (rxBufferHead + 1) & (RX_BUFFER_LEN - 1);
```

Funkce **`getReceiveData`** opět nejprve zkontroluje přítomnost dat v bufferu a v případě kladného výsledku vyčte z bufferu ukazatel na strukturu příkazu, posune konec bufferu o jeden záznam a vyčtený ukazatel vrátí:

```
pMsg = rxBuffer[rxBufferTail]; // read the message pointer
rxBufferTail = (rxBufferTail + 1) & (RX_BUFFER_LEN - 1);

return pMsg;
```

Použití kruhového bufferu je v našem kontextu výhodné, neboť v případě, že by aplikace nestačila zpracovávat přijatá data, budou staré příkazy postupně zapomínány – přepisovány novými a aktuálními – místo toho, aby došlo k přeplnění bufferu a zastavení příjmu aktuálních příkazů.

5.2. Přijímač-koordinátor

Zařízení, které jsem nazval „přijímač-koordinátor“, se stará o vytvoření sítě, příjem požadavků na asociaci koncových zařízení do této sítě, a přeposílání aplikačních příkazů z vysílače do ostatních (výkonových) koncových zařízení. Mimo vlastní vysílání musí být stále na příjmu.

Díky tomu musí implementovat volání a obsluhu příslušných primitiv rozhraní MLME a MCPS.

Každé fázi procesu vytvoření sítě odpovídá jeden stav aplikačního stavového automatu:

- `stateInit` popsaný v kapitole 5.1.1. Řízení běhu programu inicializuje použité knihovny a přejde do dalšího stavu.

- `stateScanEdStart` zavolá funkci `App_StartScan` s parametrem `gScanModeED_c`, který určuje typ scanu – ED. Pokud funkce vrátí návratovou hodnotu `errorNoError`, přejde do dalšího stavu.
- `stateScanEdWaitConfirm` – v tomto stavu se čeká, dokud ukazatel `pMsgIn` neukazuje na zprávu typu `gNwkScanCnf_c` (potvrzení MLME scanu). V takovém případě ji předá funkci `App_HandleScanEdConfirm` a přejde do dalšího stavu.
- `stateStartCoordinator` zodpovídá za odeslání požadavku na vytvoření sítě prostřednictvím funkce `App_StartCoordinator`. V případě úspěšného provedení této funkce je proveden přechod do dalšího stavu.
- `stateStartCoordinatorWaitConfirm` čeká na příchod zprávy typu `gNwkStartCnf_c` signalizující úspěšné dokončení procesu vytvoření sítě. Je pak možné přejít do posledního stavu.
- `stateListen` je stavem, ve kterém po vytvoření sítě aplikace již zůstává. Příchozí zprávy jsou obsluhovány funkcí `App_HandleMlmeInput`, která podle jejich typu volá příslušné rutiny. Dále je v každé iteraci nekonečného cyklu funkcí `handleData` vyhodnocen přijímací buffer aplikačních příkazů.

5.2.1. Interakce s vrstvou MAC

Volání a vyhodnocení primitiv rozhraní MLME zajišťují funkce zmíněné ve výčtu aplikačních stavů. Popíšu je nyní podrobněji.

Funkce **`App_StartScan`** vytváří zprávu typu `gMlmeScanReq_c`, což je požadavek na provedení 802.15.4 MAC primitiva MLME-SCAN.Request. Parametrem funkce je požadovaný typ scanu. Standard 802.15.4 definuje čtyři typy scanů, jejichž stručnou charakteristiku zde uvedu:

- *Energy Detection Scan* – během scanování je na každém kanálu po dobu zvoleného časového intervalu měřena úroveň energie produkované ostatními zařízeními v okolí. Pro každý kanál je pak vrácena hodnota v rozsahu `0x00` až `0xA0` odpovídající úrovním -80 dBm až 0 dBm.

- *Active Scan* – aktivní scan; zařízení vyšle pro každý kanál požadavek na *Beacon Frame* a po určenou dobu čeká na odpověď od potenciálních koordinátorů.
- *Passive Scan* – pasivní scan; podobně jako aktivní scan čeká na každém kanálu na *Beacon Frame*, ovšem bez předchozího *Beacon Request* příkazu.
- *Orphan Scan* – zjištění přítomnosti „osiřelého“ (*orphaned*) zařízení (*Orphaned Device* je zařízení, které není asociováno k žádné síti).

Do alokovaná zprávy se vyplní informace o typu scanu, kanálech, které chceme prozkoumat délce časového intervalu, po který má zařízení na každém kanálu naslouchat:

```
pMsg = MSG_AllocType(mlmeMessage_t);
pMsg->msgType = gMlmeScanReq_c;
pScanReq = &pMsg->msgData.scanReq;
pScanReq->scanType = scanType;
pScanReq->scanChannels[0] = (uint8_t)((SCAN_CHANNELS)     & 0xFF);
pScanReq->scanChannels[1] = (uint8_t)((SCAN_CHANNELS>>8)  & 0xFF);
pScanReq->scanChannels[2] = (uint8_t)((SCAN_CHANNELS>>16) & 0xFF);
pScanReq->scanChannels[3] = (uint8_t)((SCAN_CHANNELS>>24) & 0xFF);
pScanReq->scanDuration = 5;
```

Délka trvání scanu je určena číslem v intervalu <0; 14>, které lze přepočítat na dobu v sekundách podle vztahu:

$$t = \frac{16 \cdot 960 \cdot (2^{scanDuration} + 1)}{1000000} \text{ s}$$

Zpráva je následně přidána do fronty MLME zpráv:

```
MSG_Send(NWK_MLME, pMsg)
```

Podle úspěšnosti průběhu vykonávání vrací funkce návratové hodnoty `errorNoError` (bez chyb), `errorInvalidParameter` (špatný parametr funkce) nebo `errorAllocFailed` (nepodařilo se v paměti alokovat prostor pro zprávu).

Na funkci `App_StartScan` zmíněnou výše navazuje logicky funkce **`App_HandleScanEdConfirm`**, která vyhodnocuje výsledky získané provedením ED scanu. Ze zprávy typu `gNwkScanCnf_c` extrahuje seznam kanálů a jim odpovídajících energetických úrovní:

```
pEdList = pMsg->msgData.scanCnf.resList.pEnergyDetectList;
```

a v cyklu nalezne kanál, na němž je úroveň okolního vyzařování nejmenší:

```
for (n=0; n<16; n++)
{
     if (pEdList[n] < minEnergy)
     {
          minEnergy = pEdList[n];
          /* Channel numbering is 11 to 26 both inclusive */
          logicalChannel = n + 11;
     }
}
```

Proměnná `logicalChannel` je globální a v ní uložené číslo zvoleného kanálu je posléze použito při startu sítě. Seznam kanálů pak musí být explicitně uvolněn z paměti:

```
MSG_Free(pEdList);
```

Vlastní start koordinátoru, potažmo vytvoření sítě, je iniciováno v příslušném stavu aplikačního automatu voláním funkce **`App_StartCoordinator`**. V ní je třeba nejprve alokovat zprávu rozhraní MLME:

```
pMsg = MSG_AllocType(mlmeMessage_t);
```

V případě úspěšné alokace (ukazatele `pMsg` různého od hodnoty `NULL`) se pokračuje nastavením potřebných atributů MAC PIB – krátké adresy koordinátoru:

```
pMsg->msgType = gMlmeSetReq_c;
pMsg->msgData.setReq.pibAttribute = gMacPibShortAddress_c;
pMsg->msgData.setReq.pibAttributeValue = (uint8_t *)shortAddress;
ret = MSG_Send(NWK_MLME, pMsg);
```

a nastavení příznaku *Association Permit* řídicího přijímání asociačních požadavků:

```
pMsg->msgType = gMlmeSetReq_c;
pMsg->msgData.setReq.pibAttribute = gMacPibAssociationPermit_c;
boolFlag = TRUE;

pMsg->msgData.setReq.pibAttributeValue = &boolFlag;
ret = MSG_Send(NWK_MLME, pMsg);
```

Vzhledem k tomu, že zprávy nastavující PIB atributy nejsou řazeny do fronty (atributy jsou nastavovány ihned a není třeba čekat ve frontě na jejich vykonání), není třeba pro každý parametr alokovat novou zprávu.

Po nastavení všech potřebných parametrů je zprávě nastaven nový typ:

```
pMsg->msgType = gMlmeStartReq_c;
```

a do pomocné proměnné přiřazen ukazatel na kontejner typu `mlmeStartReq_t` uchovávající údaje potřebné pro spuštění nové PAN:

```
pStartReq = &pMsg->msgData.startReq;
```

Do tohoto kontejneru je pak zkopírován identifikátor sítě, logický kanál zvolený scanem, nastavení *Beacon Frames* (v našem případě `0x0F` – vypnuto), příznak koordinátoru sítě, příznak bateriové správy napájení, příznak reorganizace struktury sítě a povolení zabezpečení:

```
memcpy(pStartReq->panId, (void *)panId, 2);
pStartReq->logicalChannel = logicalChannel;
pStartReq->beaconOrder = 0x0F;
pStartReq->superFrameOrder = 0x0F;
pStartReq->panCoordinator = TRUE;
pStartReq->batteryLifeExt = FALSE;
pStartReq->coordRealignment = FALSE;
pStartReq->securityEnable = FALSE;
```

Nakonec je kompletní zpráva odeslána:

```
MSG_Send(NWK_MLME, pMsg);
```

I tato funkce vrací podle průběhu procesu návratové kódy `errorNoError`, `errorInvalidParameter` nebo `errorAllocFailed`.

Patřičnou reakci zařízení na příchod systémových zpráv různého typu zajišťuje funkce **`App_HandleMlmeInput`** spouštěná v každé iteraci hlavního cyklu. V případě koordinátorského zařízení obsluhuje dva typy zpráv:

- `gNwkAssociateInd_c` – indikace požadavku koncového zařízení na asociaci k síti (primitivum MLME-ASSOCIATE.Indication). Je třeba odpovědět – použít primitivum MLME-ASSOCIATE.Response. Tuto funkčnost zajišťuje funkce `App_SendAssociateResponse` popsaná dále.
- `gNwkCommStatusInd_c` – indikace úspěšně dokončené asociace koncového zařízení k síti. Není ji třeba nijak ošetřovat.

Zmíněná funkce **`App_SendAssociateResponse`** alokuje zprávu typu `gMlmeAssociateRes_c`. Ze zprávy typu `gNwkAssociateInd_c` obdržené

v parametru získá informace o zařízení, které žádá o asociaci k síti:

```
devType = pMsgIn->msgData.associateInd.deviceAddress[1] & 0x03;
```

Podle zjištěného druhu zařízení je pak kód větven – je-li koncovým zařízením přijímač, je z nejnižšího bytu adresy zjištěn osvětlovací kanál a unikátní identifikace přijímače v rámci kanálu:

```
devId = pMsgIn->msgData.associateInd.deviceAddress[0] & 0x0F;
devChannelId = ((pMsgIn->msgData.associateInd.deviceAddress[0] &
     0xF0) >> 4);
```

Je-li zařízením vysílač, je pouze zjištěn jeho identifikátor, vysílače nejsou organizovány do kanálů:

```
devId = pMsgIn->msgData.associateInd.deviceAddress[0];
devChannelId = 0;
```

Do pomocné proměnné je dále uložen ukazatel na strukturu odpovědi typu `mlmeAssociateRes_t`:

```
pAssocRes = &pMsg->msgData.associateRes;
```

Dále je nutné zkontrolovat, zda zařízení podporuje krátké adresování. Projde-li zařízení touto kontrolou, je mu přiřazena krátká adresa (první dva byty jeho rozšířené adresy) a asociace je potvrzena:

```
if (pMsgIn->msgData.associateInd.capabilityInfo &
     gCapInfoAllocAddr_c)
{
     pAssocRes->assocShortAddress[0] =
          pMsgIn->msgData.associateInd.deviceAddress[0];
     pAssocRes->assocShortAddress[1] =
          pMsgIn->msgData.associateInd.deviceAddress[1];
     pAssocRes->status = gSuccess_c;
}
```

V případě negativního výsledku je koncovému zařízení přiřazena adresa `0xFFFE` a požadavek na asociaci je zamítnut:

```
else
{
     pAssocRes->assocShortAddress[0] = 0xFE;
     pAssocRes->assocShortAddress[1] = 0xFF;
     pAssocRes->status = gPanAccessDenied_c;
}
```

Zpráva je doplněna o rozšířenou adresu koncového zařízení požadujícího asociaci a je zakázáno zabezpečení:

```
memcpy(pAssocRes->deviceAddress,
     pMsgIn->msgData.associateInd.deviceAddress, 8);
pAssocRes->securityEnable = FALSE;
```

V případě úspěšného odeslání zprávy, povolené asociaci a správného typu zařízení je nově asociované zařízení přidáno do mapy asociovaných přijímačů příslušného kanálu:

```
if (MSG_Send(NWK_MLME, pMsg) == gSuccess_c)
{
     if (requestStatus == gSuccess_c)
     {
          if (devType == devRecEnd)
          {
               associatedRecs[devChannelId] |= (1 << devId);
          }
     }
}
```

Návratové hodnoty funkce jsou `errorNoError`, `errorInvalidParameter` a `errorAllocFailed`.

Datové rozhraní MCPS je v nekonečné smyčce aplikace ošetřeno následujícím kódem:

```
if (state == stateListen)
{
     if (MSG_Pending(&mMcpsNwkInputQueue))
     {
          pMsgIn = MSG_DeQueue(&mMcpsNwkInputQueue);
          App_HandleMcpsInput(pMsgIn);
          MSG_Free(pMsgIn);
     }

     App_TransmitData();
}
```

Je-li ve frontě nalezena čekající zpráva, je ukazatel na ní načten do proměnné `pMsgIn` a předán funkci `App_HandleMcpsInput`. Poté je zpráva uvolněna z paměti. Funkce `App_TransmitData` pak zkontroluje, není-li ve výstupním bufferu připraven na odeslání nějaký aplikační příkaz.

Funkce **`App_HandleMcpsInput`** je podobná funkci `App_HandleMlmeInput` – obsluhu přijaté zprávy větví podle jejího typu. Ošetřit je třeba dva typy zpráv:

- `gMcpsDataCnf_c` – potvrzení úspěšného odeslání datové zprávy. Je dekrementován čítač zpráv čekajících ve frontě na odeslání.

- `gMcpsDataInd_c` – indikace přijaté datové zprávy. Pomocné funkci `setReceiveData` provádějící zařazení přijatých dat do přijímacího kruhového bufferu je předán ukazatel na první byte a počet přijatých bytů.

O odesílání aplikačních příkazů se stará funkce **`App_TransmitData`**. Ta se nejprve pokusí získat data z odesílacího kruhového bufferu. Je-li délka získaných dat větší než nula, pokračuje algoritmus zjištěním identifikátoru osvětlovacího kanálu z prvního bytu dat:

```
devChannelId = *(dataBuffer);
```

Podle něj je zjištěna mapa asociovaných zařízení:

```
devList = associatedRecs[devChannelId];
```

Mapa je procházena cyklem a kontrolována přítomnost asociovaného zařízení:

```
for (i = 0; i < 16 && devList > 0; i++)
{
     if ((devList >> i) & 0x01)
     {
```

Je-li v případě nalezení asociovaného zařízení splněna podmínka počtu čekajících zpráv, je alokována nová zpráva:

```
if ((numPendingPackets < MAX_PENDING_DATA_PACKETS) &&
     (pPacket == NULL))
{
     pPacket = MSG_Alloc(sizeof(nwkToMcpsMessage_t) - 1 +
          DEFAULT_DATA_LENGTH);
```

Po úspěšné alokaci je sestavena krátká adresa zařízení, která je spolu s daty zkopírována do vytvářené zprávy:

```
devShortAddress[0] = (uint8_t) ((devChannelId & 0x0F) << 4) | i;

pPacket->msgType = gMcpsDataReq_c;
memcpy(pPacket->msgData.dataReq.msdu, (void *) dataBuffer,
     msduLength);
memcpy(pPacket->msgData.dataReq.dstAddr, devShortAddress, 2);
memcpy(pPacket->msgData.dataReq.srcAddr, (void *)shortAddress, 2);
memcpy(pPacket->msgData.dataReq.dstPanId, (void *) panId, 2);
memcpy(pPacket->msgData.dataReq.srcPanId, (void *) panId, 2);

pPacket->msgData.dataReq.dstAddrMode = gAddrModeShort_c;
pPacket->msgData.dataReq.srcAddrMode = gAddrModeShort_c;
pPacket->msgData.dataReq.msduLength = msduLength;
```

Poté jsou nastaveny parametry zprávy (přímý přenos, bez potvrzování, vypnuté

zabezpečení) a zpráva je odeslána:

```
pPacket->msgData.dataReq.txOptions = gTxOptsAck_c;
pPacket->msgData.dataReq.msduHandle = msduHandle++;

NR MSG_Send(NWK_MCPS, pPacket);
```

V provozním stavu `stateListen` je v průběhu každé iterace hlavního cyklu volána funkce **`handleData`** vyhodnocující přijímací buffer aplikačních příkazů. Do proměnné `pMsg` se na začátku pokusí získat ukazatel na nově přijatý příkaz:

```
pMsg = getReceiveData(); // get incoming data from the buffer
```

Tato proměnná je následně testována na hodnotu `NULL` signalizující prázdný buffer. V případě nenulové hodnoty pokračuje algoritmus zjištěním, zda je příkaz určen pro osvětlovací kanál, ke kterému koordinátor náleží. V takovém případě předá zprávu vyhodnocovací rutině `handleMessage`:

```
if (pMsg->channelId == ((shortAddress[0] & 0xF0) >> 4))
{
     handleMessage(pMsg);
}
```

V každém případě ovšem zprávu zařadí do odesílacího bufferu, neboť je třeba ji předat všem koncovým zařízením náležejícím specifikovanému kanálu:

```
setTransmitData(pMsg);
```

Zmíněná funkce **`handleMessage`** analyzuje přijatý příkaz. Jeho obsluha se větví podle identifikátoru příkazu (viz kapitola 3.1. Komunikační protokol). Příkazy proporcionální změny jasu svítidla pracují s globální proměnnou `outputCmpVal` udržující aktuální nastavení hodnoty *Output Compare* registru druhého kanálu časovače TPM1. Při dosažení této hodnoty čítačem je vyvoláno přerušení.

Smysluplný rozsah nastavení hodnot tohoto registru je 0 až 1250. Číslo 1250 je totiž nejvyšší hodnotou, kterou čítač dosáhne za 10 ms (půlperiodu síťového napětí):

$$\frac{16\cdot 10^{6}\,\mathrm{Hz}}{128}\cdot 10\cdot 10^{-3}\,\mathrm{s}=1250$$

V tomto výrazu je 16 MHz frekvencí hodinového signálu mikrokontroléru, 128 je hodnota předděličky čítače TMP1 a 10 ms délka intervalu čítání.

Hodnota proměnné `outputCmpVal` je pak měněna o součin parametru příkazu a čísla 12 (přibližně setiny hodnoty 1250), čímž je dán převod procentuálního vyjádření parametru příkazu na přírůstek/úbytek hodnoty proměnné `outputCmpVal`:

```
case 0x01: // increase
     outputCmpVal -= pMsg->data * 12; // data - percentage
     ZC_ENABLE;
     break;

case 0x02: // decrease
     outputCmpVal += pMsg->data * 12;
     ZC_ENABLE;
     break;
```

Makro `ZC_ENABLE` povoluje přerušení modulu KBI (detekce průchodu nulou). Než je aktualizovaná hodnota proměnné `outputCmpVal` zapsána do registru časovače, je provedeno omezení její hodnoty na interval <0; 1250>:

```
if (outputCmpVal > 1250) outputCmpVal = 1250;
if (outputCmpVal < 0) outputCmpVal = 0;

TPM1C2VL = (uint8_t) ((outputCmpVal)      & 0xFF);
TPM1C2VH = (uint8_t) ((outputCmpVal >> 8) & 0xFF);
```

V případě přijetí příkazů skokového zapnutí/vypnutí svítidla je proměnná `outputCmpVal` modifikována také (nastavena na jednu ze svých krajních hodnot), z důvodu šetření systémovými prostředky jsou však zakázána přerušení modulů KBI i TPM1 a výkonový výstup je zapnut/vypnut trvale:

```
case 0x03: // switch on
     outputCmpVal = 0;
     OUTPUT_ON;
     ZC_DISABLE;
     TPM_DISABLE;
     break;

case 0x04: // switch off
     outputCmpVal = 1250;
     OUTPUT_OFF;
     ZC_DISABLE;
     TPM_DISABLE;
     break;
```

5.2.2. Rutiny řízení hardware

Během inicializačního stavu aplikace je jako poslední volána funkce **`HW_init`** obsahující rutiny počátečního nastavení periferií mikrokontroléru, které jsou použity pro fázové řízení spotřebiče a signalizaci provozních stavů.

Je zde nastaven pin s LED jako výstupní a LED je zhasnuta (pin nastaven do 1):

```
LED_DD |= LED_PIN;
LED_PE &= ~LED_PIN;
LED_P  |= LED_PIN;
```

Dále je na detekci hrany inicializována periferie KBI, jejíž přerušení je použito ve spojení s obvodem detekce průchodu síťového napětí nulou:

```
ZC_DISABLE; // KBIE=0
KBIPE = 0x80; // KBIPE7=1
ZC_CLEAR; // KBACK=1
```

Pro generování zapínacích impulsů triaku ve správné vzdálenosti od průchodu síťového napětí nulou (dané řídicím úhlem) je použit kanál CH2 časovače TPM1 (zdroj hodin – *System Bus*, předdělička 128, *Output Compare Interrupt*):

```
TPM1SC   = 0x0F;
TPM1MODL = 0xFF;
TPM1MODH = 0xFF;

// software only output compare on channel 2
TPM1C2SC = 0x10;
TPM1C2VL = 0xFF;
TPM1C2VH = 0xFF;
```

Optotriak ovládající výkonový triak je připojen na pin PD2 (v našem případě z důvodu snadné modifikovatelnosti kódu definovaný pomocí maker preprocesoru):

```
OUTPUT_DD |= OUTPUT_PIN;
OUTPUT_PE &= ~OUTPUT_PIN;
OUTPUT_P  |= OUTPUT_PIN;
```

Přerušení modulu KBI (neboli signalizace průchodu síťového napětí nulou) je obslouženo rutinou **`KBI_ISR`**. Na jejím začátku je smazán příznak přerušení a obrácen směr detekované hrany:

```
ZC_CLEAR;
ZC_TOGGLE;
```

Dále je aktualizována proměnná `outputState` udržující stav výstupního pinu:

```
outputState = 0;
```

a výstupní pin je pomocí makra `OUTPUT_OFF` nastaven do logické jedničky:

```
OUTPUT_OFF;
```

Na závěr je (opět pomocí maker definovaných na začátku aplikačního souboru) vynulován čítač TPM1 a povoleno jeho *Output Compare* přerušení:

```
TMP_CLEAR;
TPM_ENABLE;
```

Na tuto funkci logicky navazuje obsluha přerušení kanálu CH2 čítače TPM1 - **`TPM1CH2_ISR`**. Na začátku je klasicky vynulován příznak přerušení:

```
TMP_CLEARF;
```

Podle stavu výstupního pinu v proměnné `outputState` je pak obsluha rozdělena na dva případy:

- `outputState == 0` – jde o první volání obsluhy v rámci jedné půlperiody síťového napětí. Je třeba vygenerovat zapínací impuls triaku:

```
cmpVal = (uint16_t) (outputCmpVal + 20);
if (cmpVal > 1250) cmpVal = 1250;

TPM1C2VL = (uint8_t) ((cmpVal)      & 0xFF);
TPM1C2VH = (uint8_t) ((cmpVal >> 8) & 0xFF);
outputState = 1;
OUTPUT_ON;
```

  Délku impulsu jsem empiricky definoval jako $20/1250 \cdot 10\ \text{ms} = 160\ \mu\text{s}$ (10 ms je délka jedné půlperiody síťového napětí o frekvenci 50 Hz, 1250 je hodnota, které časovač nastavený dle popisu výše dosáhne za 10 ms).

- `outputState == 1` – druhé volání v rámci půlperiody, neboli konec zapínacího impulsu. Je třeba zakázat toto přerušení a nastavit na výstupu log. jedničku:

```
TPM_DISABLE;
outputState = 0;
OUTPUT_OFF;
```

O řízení dalšího HW (rozhraní SPI a některé I/O piny) se stará knihovna *802.15.4 MAC PHY* sama – obsluha je součástí fyzické vrstvy.

5.3. Vysílač

Vysílač je zařízení, které jako jediné z navržených druhů zařízení přichází do přímého styku s uživatelem.

Stavový automat řídicí činnost zařízení má tyto stavy:

- `stateInit` popsaný v kapitole 5.1.1. Řízení běhu programu inicializuje použité

knihovny a přejde do dalšího stavu.

- `stateScanActiveStart` – je volána funkce `App_StartScan` s parametrem `gScanModeActive_c`, čímž je provedena žádost o aktivní scan. Je-li návratová hodnota `errorNoError`, skočí automat do dalšího stavu.
- `stateScanActiveWaitConfirm` – čeká se nejprve na příchod zprávy typu `gNwkBeaconNotifyInd_c`, která signalizuje příchod *Beacon Frame*. Je-li přijata, je pouze uvolněna z paměti. Poté se čeká na příchod zprávy typu `gNwkScanCnf_c`, která nese informaci o proběhlém aktivním scanu. Tato zpráva je následně předána funkci `App_HandleScanActiveConfirm`, která ji analyzuje a vrátí výsledek hledání vhodné sítě PAN. V případě kladného výsledku je postoupeno do dalšího stavu, v případě neúspěchu je automat vrácen do stavu předchozího.
- `stateAssociate` volá funkci `App_SendAssociateRequest`, která dříve nalezenému koordinátoru zašle požadavek o asociaci k síti. Poté dojde k přechodu na další stav automatu.
- `stateAssociateWaitConfirm` – v tomto stavu se opět čeká na příchod zprávy, tentokrát typu `gNwkAssociateCnf_c`. Obsahuje informace o povolení asociace k síti a přidělenou krátkou adresu. Je zpracována funkcí `App_HandleAssociateConfirm`. Tímto krokem asociace k síti končí a je proveden přechod do dalšího stavu.
- `stateListen` je stavem, ve kterém po asociaci k síti aplikace již zůstává. Příchozí zprávy jsou obsluhovány funkcí `App_HandleMlmeInput`, která podle jejich typu volá příslušné rutiny. Dále je v každé iteraci nekonečného cyklu funkcí `handleInputs` obsloužen vstup od uživatele (tedy klávesnice a scroller).

5.3.1. Interakce s vrstvou MAC

Funkce **`App_StartScan`**, která má na starosti zahájení aktivního scanování okolí, je identická se stejně pojmenovanou funkcí popsanou v kapitole 5.2. Přijímač-

koordinátor. Jedinou změnou je její volání – jak již bylo zmíněno, volá se s parametrem `gScanModeActive_c`, který zabezpečí volbu správné metody scanu.

Výsledek scanu je pak zpracován funkcí **`App_HandleScanActiveConfirm`**. Ta projde seznam nalezených koordinátorů a zvolí takový, jehož síť má námi požadované PAN ID a jehož signál je ze skupiny koordinátorů se stejným PAN ID nejsilnější. Podmínkou volby koordinátoru je také jeho ochota asociovat nová koncová zařízení do sítě a podpora komunikace bez použití *Beacon Frames*. Informace o nalezeném koordinátoru jsou pak zkopírovány do globální proměnné `coordInfo` typu `panDescriptor_t`:

```
if((pPanDesc->superFrameSpec[1] & gSuperFrameSpecMsbAssocPermit_c)
 &&((pPanDesc->superFrameSpec[0] & gSuperFrameSpecLsbBO_c)==0x0F))
{
     if (pPanDesc->coordPanId[0] == panId[0] &&
          pPanDesc->coordPanId[1] == panId[1] &&
          pPanDesc->linkQuality > bestLinkQuality)
     {
          memcpy(&coordInfo, pPanDesc, sizeof(panDescriptor_t));
          bestLinkQuality = pPanDesc->linkQuality;
          rc = errorNoError;
     }
}
```

Seznam koordinátorů je nakonec uvolněn z paměti.

V dalším stavu aplikace je žádáno o asociaci do vybrané sítě. Tato funkčnost je zapouzdřena funkcí **`App_SendAssociateRequest`**, která MLME zprávou typu `gMlmeAssociateReq_c` volá MAC primitivum MLME-ASSOCIATE.Request. Součástí této zprávy je datový kontejner typu `mlmeAssociateReq_t` obsahující informace o zvoleném koordinátoru zkopírované z globální proměnné `coordInfo`:

```
pAssocReq = &pMsg->msgData.associateReq;

memcpy(pAssocReq->coordAddress, coordInfo.coordAddress, 8);
memcpy(pAssocReq->coordPanId,   coordInfo.coordPanId, 2);
pAssocReq->coordAddrMode      = coordInfo.coordAddrMode;
pAssocReq->logicalChannel     = coordInfo.logicalChannel;
pAssocReq->securityEnable     = FALSE;
pAssocReq->capabilityInfo     = gCapInfoAllocAddr_c;
```

Konstanta `gCapInfoAllocAddr_c` přiřazená parametru `capabilityInfo` znamená požadavek na přiřazení krátké adresy.

V případě úspěchu vrací funkce návratovou hodnotu `errorNoError`, chybové stavy jsou opět popsány kódy `errorInvalidParameter` a `errorAllocFailed`.

Výsledek asociace vyhodnocuje funkce **`App_HandleAssociateConfirm`**. V případě, že je koncovému zařízení koordinátorem přiřazena rezervovaná krátká adresa `0xFFFE`, znamená to, že zařízení nebyla přidělena validní krátká adresa. V kontextu mé aplikace se jedná o zamítnutí asociace zařízení do sítě. Avšak v případě, že je požadavek na asociaci sestaven dle postupu uvedeného výše, neměla by tato eventualita nastat. Zařízení v takovém případě musí používat rozšířený adresační režim:

```
if ((pMsg->msgData.associateCnf.assocShortAddress[0] >= 0xFE) &&
     (pMsg->msgData.associateCnf.assocShortAddress[1] == 0xFF) )
{
     myAddrMode = gAddrModeLong_c;
     memcpy(myAddress, (void *) aExtendedAddress, 8);
}
```

Dopadla-li asociace úspěšně, je uložena získaná krátká adresa a zvolen patřičný adresační mód:

```
else
{
     myAddrMode = gAddrModeShort_c;
     memcpy(myAddress,
          pMsg->msgData.associateCnf.assocShortAddress, 2);
}
```

V každé iteraci aplikačního cyklu je spouštěna funkce **`App_HandleMlmeInput`** ošetřující reakci na systémové zprávy různého typu během provozního stavu aplikace. V případě koncových zařízení je její jedinou úlohou reagovat na MLME zprávu typu `gNwkPollCnf_c` potvrzující provedení *Data Poll* (žádosti o data). Je zjištěno, zda byl požadavek úspěšný, a v záporném případě je nastavena menší frekvence žádostí:

```
if (pMsg->msgData.pollCnf.status != gSuccess_c)
{
     // slow down polling
     pollInterval = POLL_INTERVAL_SLOW;
     // do not wait for poll confirm
     waitPollConfirm = FALSE;
}
```

Podobná funkce **`App_HandleMcpsInput`** obsluhuje datové zprávy. Podporovanými typy zpráv jsou:

- `gMcpsDataCnf_c` – potvrzení odeslání datového rámce MAC vrstvou. Je dekrementován čítač čekajících zpráv `numPendingPackets`.
- `gMcpsDataInd_c` – signalizace obdrženého datového rámce. Je zvýšena frekvence *pollingu*.

O odesílání dat nashromážděných ve frontě aplikačních příkazů se stará funkce **`App_TransmitData`**. V rámci svého provádění nejprve zkontroluje, zda je možné alokovat novou datovou zprávu (není-li počet čekajících zpráv větší než nastavený nebo není-li již prázdná zpráva alokována). V kladném případě zprávu alokuje a ukazatel na ni umístí do globální proměnné `pPacket`. Navazující kód tuto proměnnou kontroluje na hodnotu `NULL`. Dále je učiněn pokus o přečtení dat z odesílacího kruhového bufferu – do proměnné `msduLength` je uložena návratová hodnota funkce `getTransmitData`. Je-li větší nuly, pokračuje se naplněním zprávy:

```
if (pPacket != NULL)
{
     uint8_t msduLength =
          getTransmitData(pPacket->msgData.dataReq.msdu);

     if (msduLength)
     {
```

Zprávě je nastaven typ `gMcpsDataReq_c` a jsou do ní zkopírována adresní data (o kopii užitečných dat do bufferu zprávy se postarala funkce `getTransmitData`):

```
pPacket->msgType = gMcpsDataReq_c;
memcpy(pPacket->msgData.dataReq.dstAddr,
     coordInfo.coordAddress, 8);
memcpy(pPacket->msgData.dataReq.srcAddr, myAddress, 8);
memcpy(pPacket->msgData.dataReq.dstPanId,
     coordInfo.coordPanId, 2);
memcpy(pPacket->msgData.dataReq.srcPanId, coordInfo.coordPanId,
     2);
pPacket->msgData.dataReq.dstAddrMode = coordInfo.coordAddrMode;
pPacket->msgData.dataReq.srcAddrMode = myAddrMode;
pPacket->msgData.dataReq.msduLength = msduLength;
```

Je nastaven požadavek na *Acknowledge* a zpráva je odeslána:

```
pPacket->msgData.dataReq.txOptions = gTxOptsAck_c;
NR MSG_Send(NWK_MCPS, pPacket);
```

Po odeslání se ukazatel `pPacket` nastaví na hodnotu `NULL` a inkrementuje čítač čekajících zpráv `numPendingPackets`.

Tímto výčet komunikačních rutin končí, jako rezerva pro případné budoucí rozšíření funkčnosti systému jsou v kódu připraveny funkce pro nepřímý přenos dat z koordinátorského zařízení na koncové zařízení (`App_ReceiveData`).

5.3.2. Rutiny řízení hardware

Při vývoji firmware vysílače jsem narazil na omezení – jeden ze zapůjčených modulů měl poškozeny některé z I/O pinů, a to bohužel i v rámci portu A, který je plně obsazen a není tak k dispozici rezervní pin pro případ problémů tohoto charakteru. Poškození se týkalo i obvodu přerušení modulu KBI, který nebyl schopen generovat přerušení při hraně vstupního signálu. Obsluhu klávesnice a scrolleru jsem tak musel implementovat metodou *pollingu*, neboli periodickým čtením stavu vstupních pinů. Nebylo tak možné mezi jednotlivými příkazy uvést zařízení do některého z režimů spánku, což se nepříznivě projevilo na spotřebě.

O inicializaci mnou používaných periferií se stará funkce **`HW_init`**. Nejprve je nastaven výstupní pin s připojenou indikační LED:

```
LED_DD |= LED_PIN;
LED_PE &= ~LED_PIN;
LED_P  |= LED_PIN;
```

Vstupní prvky (klávesnice, scroller) jsou připojeny ke vstupnímu portu A a dvěma výstupním pinům portu B (v kódu zastupovaným makry):

```
KBD_BTN_DD = 0x00;
KBD_BTN_PE = 0x00;
KBD_EN_DD |= (KBD_EN1 | KBD_EN2);
KBD_EN_PE &= ~(KBD_EN1 | KBD_EN2);
KBD_EN_P  |= (KBD_EN1 | KBD_EN2);
```

Vlastní obsluhu provádí funkce **`handleInputs`** volaná v provozním stavu každou iteraci nekonečného cyklu. Pro udržování informací o stavu vstupů používá globální proměnné `kbdState` a `ircState`. Z nich a aktuálního stavu uloženého v lokálních proměnných `actualKbdState` a `actualIrcState` je pak algoritmus schopen detekovat změnu od minulého stavu. Logickou funkcí XOR je

z proměnných `kbdState` a `actualKbdState` odvozen obsah proměnné `diff` – změna vstupního signálu je pak indikována jedničkou v této proměnné:

```
if (kbdState != PTAD)
{
     actualKbdState = PTAD;
     diff = kbdState ^ actualKbdState;
```

Podle přítomnosti jedniček na daných pozicích v proměnné `diff` je pak aktivována obsluha jednotlivých vstupních zařízení. Změna na pozicích dvou nejvyšších bitů znamená událost scrolleru:

```
if (diff & 0xC0)
{
     if (actualKbdState & IRC_CH1) actualIrcState |= 0x01;
     if (actualKbdState & IRC_CH2) actualIrcState |= 0x02;
```

Nejprve je aktualizována proměnná `actualIrcState`. Její hodnota sestává ze dvou bitů odpovídajících jednotlivých složkám signálu IRC a nabývá hodnot od nuly do čtyř. Porovnáním současné a předchozí hodnoty (uložené v `ircState`) této proměnné získáme informaci o směru otáčení scrolleru:

```
if ((ircState == 0 && actualIrcState == 1) ||
    (ircState == 1 && actualIrcState == 3) ||
    (ircState == 3 && actualIrcState == 2) ||
    (ircState == 2 && actualIrcState == 0))
{
     ircDir = 0;
}
else
{
     ircDir = 1;
}
```

Za pomoci této hodnoty již můžeme vytvořit nový aplikační příkaz (zvýšení nebo snížení intenzity světla) a přidat ho do odesílacího kruhového bufferu:

```
pMsg = MSG_AllocType(appMessage_t);
pMsg->channelId = actualChannelId;
pMsg->command = 0x01 + ircDir; // 0x01 - increase, 0x02 - decrease
pMsg->data = 0x01; // 1 percent
setTransmitData(pMsg);
```

V případě, že nastala změna na místě pátého nebo šestého bitu a zároveň je to náběžná hrana, jsou obslouženy tlačítka skokového zapnutí/vypnutí svítidla:

```
if ((diff & 0x30) && ((actualKbdState & KBD_BTN5) ||
                      (actualKbdState & KBD_BTN6)))
```

Je opět alokován aplikační příkaz, stanoven kód příkazu a příkaz zařazen do odesílacího bufferu:

```
pMsg = MSG_AllocType(appMessage_t);
pMsg->channelId = actualChannelId;

if (actualKbdState & KBD_BTN5)
{
     pMsg->command = 0x03; // 0x03 - switch on
}
else if (actualKbdState & KBD_BTN6)
{
     pMsg->command = 0x04; // 0x04 - switch off
}

pMsg->data = 0x00; // no data to send
setTransmitData(pMsg);
```

Poslední možností je, že se změnil některý ze čtyř nejnižších bitů. V takovém případě je třeba provést test maticové klávesnice (viz kapitolu 2.5.1. Maticová klávesnice). Je tedy nejprve buzena horní řada tlačítek sledován vstupní port (vše se opět děje za pomoci maker preprocesoru zastupujících konkrétní periferie):

```
KBD_REL2;
KBD_SET1;

if (~KBD_BTN_P & KBD_BTN1) actualChannelId = 0;
if (~KBD_BTN_P & KBD_BTN2) actualChannelId = 1;
if (~KBD_BTN_P & KBD_BTN3) actualChannelId = 2;
if (~KBD_BTN_P & KBD_BTN4) actualChannelId = 3;
```

Při zjištění stisknutého tlačítka je příslušné číslo kanálu uloženo do globální proměnné `actualChannelId`. Totéž je pak provedeno pro druhou řadu tlačítek:

```
KBD_REL1;
KBD_SET2;

if (~KBD_BTN_P & KBD_BTN1) actualChannelId = 4;
if (~KBD_BTN_P & KBD_BTN2) actualChannelId = 5;
if (~KBD_BTN_P & KBD_BTN3) actualChannelId = 6;
if (~KBD_BTN_P & KBD_BTN4) actualChannelId = 7;
```

Nakonec je obnoveno buzení obou řad tlačítek:

```
KBD_SET1;
KBD_SET2;
```

Funkce končí uložením obsahu proměnné `actualKbdState` do globální proměnné `kbdState` zajišťující uchování této hodnoty do dalšího volání funkce.

Jednodušší operace s periferiemi (např. ovládání LED) jsou prováděny přímo prostřednictvím maker definovaných na začátku aplikačního zdrojového souboru.

5.4. Koncový přijímač

Stavy aplikačního automatu koncového přijímače se shodují se stavy vysílače (jde také o koncové zařízení). Rozdíl spočívá v jednak v tom, že přijímač je (podobně jako koordinátor) neustále na příjmu (síťové napájení mu to bez problémů umožňuje) a sám nic nevysílá, jednak v rozdílné obsluze hardware (která je zase shodná s přijímačem-koordinátorem). Dalo by se tedy říci, že firmware koncového přijímače je kombinací firmware vysílače a koordinátoru.

5.4.1. Interakce s vrstvou MAC

Jediná významněji upravená funkce je rutina zpracování odpovědi na asociační požadavek **`App_HandleAssociateConfirm`**. Bylo do ní třeba přidat nastavení MAC PIB atributu `macRxOnWhenIdle`, který řídí přepnutí transceiveru do režimu příjmu v době nečinnosti. U zařízení koordinátorského typu je tento atribut nastaven automaticky vrstvou MAC při spuštění sítě, koncové zařízení si ho musí v případě zájmu nastavit explicitně.

Nastavení se děje standardně přes zprávu typu `gMlmeSetReq_c`. Musí tedy být nejprve alokována, posléze je třeba jí přiřadit patřičný atribut a ukazatel na jeho novou hodnotu. Pak je zpráva odeslána:

```
pMsg = MSG_AllocType(mlmeMessage_t);
pMsg->msgType = gMlmeSetReq_c;
pMsg->msgData.setReq.pibAttribute = gMacPibRxOnWhenIdle_c;
value = 0x01;
pMsg->msgData.setReq.pibAttributeValue = &value;
NR MSG_Send(NWK_MLME, pMsg);
```

Od této chvíle již zařízení očekává data od koordinátora zaslaná přímým způsobem (*Direct Data Transfer*).

5.4.2. Rutiny řízení hardware

Jak již bylo zmíněno, obsluha HW je shodná s koordinátorským zařízením – viz podkapitolu 5.2.2. Rutiny řízení hardware kapitoly 5.2. Přijímač-koordinátor.

6. Závěr

Zadání této práce stanovuje cíle v poměrně obecné rovině, které se mi podařilo prakticky v plné míře splnit. Jedinou výjimkou je realizace, kdy pro momentální nedostatek volných bezdrátových modulů byly realizovány pouze dva druhy zařízení – vysílač a přijímač-koordinátor. Zbývající koncový přijímač však sdílí hardwarovou konstrukci s přijímačem-koordinátorem – jde tedy spíše o formální záležitost.

Mimo těchto obecných cílů jsem si vytyčil několik dílčích met, kterých měla má konstrukce dosáhnout. Jednalo se mimo jiné o možnost řízení více osvětlovacích kanálů, o požadavky na mechanickou konstrukci a v neposlední řadě také o snadnou rozšiřitelnost a modularitu. Mimo požadavku na zanedbatelnou klidovou spotřebu vysílače, na který jsem z důvodů uvedených v práci musel rezignovat, se mi při implementaci systému podařilo dostát i těmto vlastním nárokům.

Možnosti vylepšení stávajícího systému jsou nasnadě – v případě dalšího vývoje směřujícího například ke komerční aplikaci by prvním krokem pravděpodobně byla náprava klidové spotřeby vysílače. Co se týče nové funkcionality, dovedu si představit výhody dynamického rozdělování koncových zařízení do kanálů, nejlépe za pomoci bezdrátové konfigurační jednotky řízené počítačem. Praktický by mohl být také návrh koncového zařízení zajišťujícího rozhraní na některý ze standardních protokolů řízení osvětlení (např. DMX512-A, X10 nebo KNX).

Tato práce mi umožnila prakticky využít a dále prohloubit znalosti základních principů moderní bezdrátové komunikace získané dosavadním studiem. Měl jsem příležitost zjistit, na jak komplexních základech stojí výsledná funkčně jednoduchá aplikace. Nezanedbatelný přínos pro mne rovněž znamená získání zkušeností s návrhem síťově napájených zařízení a s fázovou regulací výkonu spotřebiče.

7. Seznam použitých zdrojů

[1] Různí autoři. *IEEE 802.15.4*. http://en.wikipedia.org/wiki/IEEE_802.15.4

[2] Hanus, Stanislav. *Rádiové a mobilní komunikace*. Brno: Ústav radioelektroniky FEKT VUT, 2002. 85 s. ISBN nemá.

[3] Obrázek „*Topologie sítí ZigBee*“, http://cs.wikipedia.org/wiki/Soubor:Topologie.jpg

[4] Kysilka, Radek. *ZigBee*. http://www.lupa.cz/clanky/zigbee/

[5] Bradáč, Zdeněk. *Bezdrátový komunikační standard ZigBee*. http://www.automatizace.cz/article.php?a=638

[6] Různí autoři. *ZigBee*. http://cs.wikipedia.org/wiki/ZigBee

[7] Obrázek „*Modul XBee*“, http://www.digi-intl.co.jp/images/xbee.jpg

[8] Obrázek „*Modul ZigBit*“, http://www.automatedbuildings.com/news/sep07/articles/meshnetics/zigbit.jpg

[9] Obrázek „*Modul osazený PAN4450*“, http://www.automatizace.cz/images/article/1183_fhbjnj.jpg

[10] Různí autoři. *Galvanický článek*. http://cs.wikipedia.org/wiki/Galvanick%C3%BD_%C4%8Dl%C3%A1nek

[11] Vrba, Jaromír. *Výkonová elektronika 1*. Brno: Ústav výkonové elektrotechniky a elektroniky FEKT VUT, 2003. 140 s. ISBN nemá.

[12] Obrázek „*Krabička KPDO3*“, http://www.ges.cz/images/pictures/k/kpdo3.jpg

[13] Fairchild Semiconductor. *MOC3020-M*. http://www.fairchildsemi.com/ds/MO/MOC3020-M.pdf

[14] Bourns, Inc.. *TIC226*. http://www.bourns.com/pdfs/tic226.pdf

[15] Obrázek „*Krabička LK 80x28*“, http://www.elektromaterialy.cz/files/products/1%5B3%5D.jpg

[16] Freescale Semiconductor. *802.15.4 MAC PHY Software Reference Manual*. http://www.freescale.com/files/rf_if/doc/ref_manual/802154MPSRM.pdf

[17] Freescale Corporation. *802.15.4 MAC PHY Software User's Guide*. http://www.freescale.com/files/rf_if/doc/ref_manual/802154MPSUG.pdf

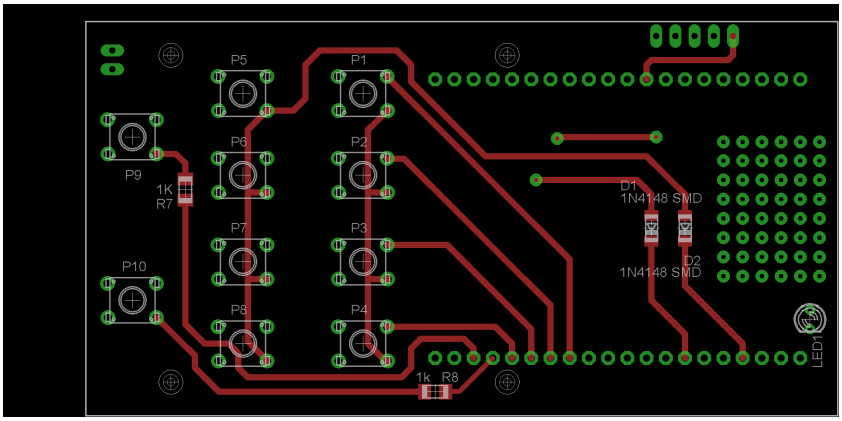
8. Seznam použitých zkratek a symbolů

ACK	Acknowledge, příznak úspěšného přijetí dat
ASP	Application Support Package, proprietární rozhraní firmy Freescale pro řízení spotřeby, přístup do NV RAM atd.
DPS	Deska plošných spojů
ED	Energy Detection, scanování s detekcí energetických úrovní
FFD	Fully Functional Device, zařízení s plnou funkčností
IRC	Incremental Rotary Encoder, inkrementální snímač otáčení
KBI	Keyboard Interrupt, periferie mikrokontrolérů řady HCS08
MAC	Medium Access Control, subverstva linkové vrstvy OSI modelu
MCPS	MAC Common Part Sublayer, datové rozhraní
MLME	MAC subLayer Management Entity, rozhraní systémových příkazů
NV	Non-volatile, pamět uchovávající data i po odpojení napájení
PAN	Personal Area Network, osobní síť
PANID	PAN Identification, identifikátor vytvořené sítě PAN
PCB	Printed Circuit Board, deska plošných spojů
PIB	PAN Information Base, soubor nastavení vrstvy MAC
PHY	PHYsical Layer, fyzická vrstva OSI modelu
RFD	Reduced Functionality Device, zařízení s omezenou funkčností
SAP	Service Access Point, styčný bod jednotlivých vrstev OSI modelu
SMD	Surface Mount Device, součástka pro povrchovou montáž
SMT	Surface Mount Technology, technologie povrchové montáže
PSI	Serial Peripheral Interface, sériové rozhraní použité pro komunikaci s transcieverem MC13193
U(S)ART	Universal (Synchronous and) Asynchronous serial Receiver and Transmitter, (synchronní a) asynchronní transceiver pro všeobecné použití
WPAN	Wireless Personal Area Network, bezdrátová osobní síť
XOR	Exclusive OR, nonekvivalence

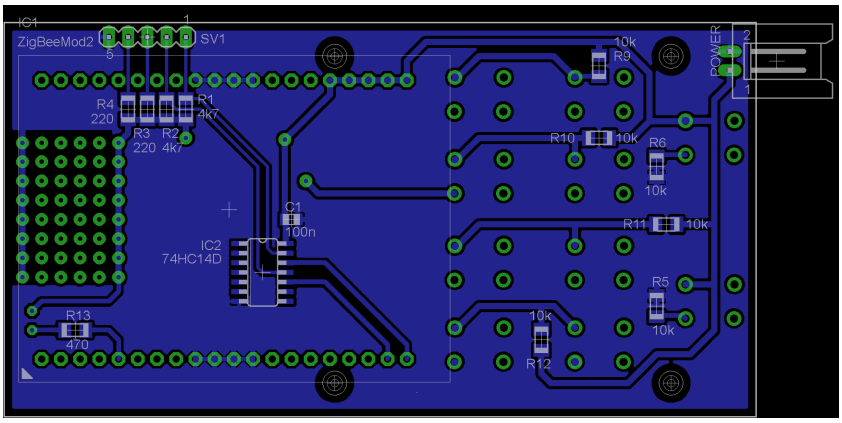
9. Přílohy

9.1. Návrh DPS

9.1.1. Vysílač

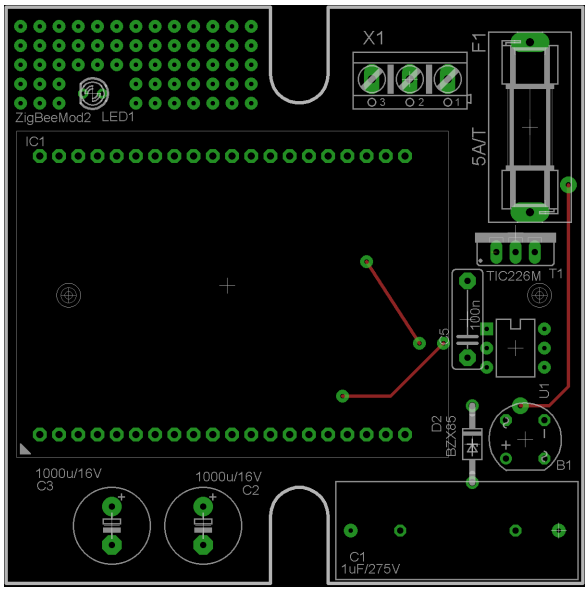


Ilustrace 9.1: Horní strana plošného spoje vysílače

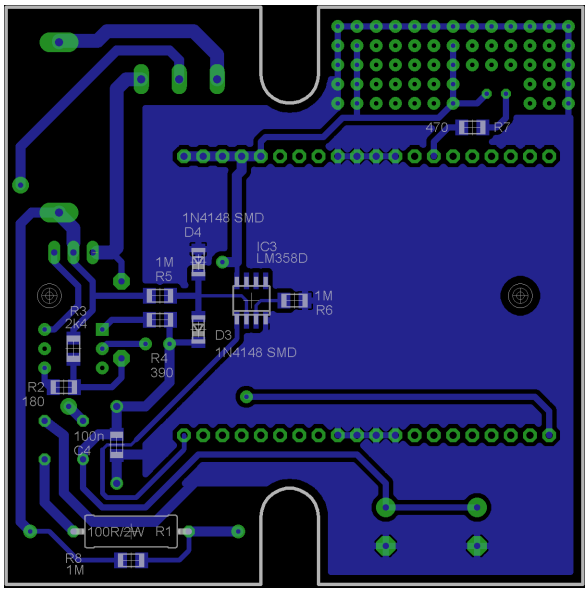


Ilustrace 9.2: Spodní strana plošného spoje vysílače

9.1.2. Přijímač



Ilustrace 9.3: Horní strana plošného spoje přijímače



Ilustrace 9.4: Spodní strana plošného spoje přijímače

9.2. Schémata zapojení

Viz následující listy.